AD-A285 472

WL-TR-94-1093

PATTERN THEORETIC KNOWLEDGE DISCOVERY

Jeffrey Alan Goldman

August 1994

Final Report for Period December 1993- August 1994

DTIC
ELECTE
OCT 13 1994
G

Approved for Public Release; Distribution is unlimited

53PA 94-32423 422730

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-7409

NOTICE

When Government drawings, specifications or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility nor any obligation whatsoever. The fact that the government may have formulated, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise in any manner construed, as licensing the holder or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

JEFFREY ALAN GOLDMAN
Computer Scientist
Technology Section

DOUGLAS S. HAGER, Chief
Technology Section
Target Recognition Technology Branch

WILLIAM E. MOORE, Acting Chief
Mission Avionics Division
Avionics Directorate

If your address has changed, if you wish to be removed from our mailing list or if the addressee is no longer employed by your organization, please notify WL/AART, Bldg 22, 2690 C Street STE 1, Wright-Patterson AFB, OH 45433-7408 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 94	Final Report Dec 93 - Aug 94

4. TITLE AND SUBTITLE
Pattern Theoretic Knowledge Discovery

5. FUNDING NUMBERS
PE 62204F
PR 0100
TA AA
WU 13

6. AUTHOR(S)
Jeffrey Alan Goldman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Avionics Directorate
Wright Laboratory
Air Force Material Command
Wright-Patterson AFB OH 45433-7409

8. PERFORMING ORGANIZATION REPORT NUMBER
WL-TR-94-1093

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Avionics Directorate
Wright Laboratory
Air Force Material Command
Wright-Patterson AFB OH 45433-7409

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
WL-TR-94-1093

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

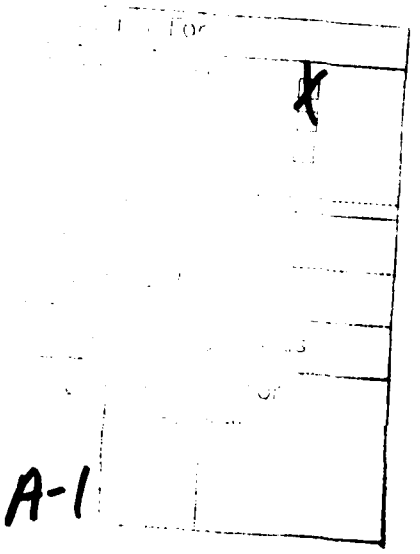
13. ABSTRACT *(Maximum 200 words)*

Among the future research directions of Knowledge Discovery in Databases is the ability to extract an overlying concept relating data objects that are useful to the investigator. Some of the current limitations involve the search complexity and what it means to be "useful." The Pattern Theory research crosses over in a natural way to the aforementioned domain. The goal of this paper is threefold. First, we wish to present a new approach to the problem of learning by Discovery and robust pattern finding in general. Second, we will show its performance by exhibiting several learning curves. Third, from a practical standpoint, we wish to explore the current limitations of a Pattern Theoretic Discovery and Databases problem. Function decomposition is the central core of Pattern Theory. The development allows us to discuss the notion of "patterns," and thus, the notion of "useful," in a formal manner.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Pattern Theory, Function Decomposition, Machine Learning Patterns, Knowledge Discovery	53
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Contents



List of Figures

List of Tables

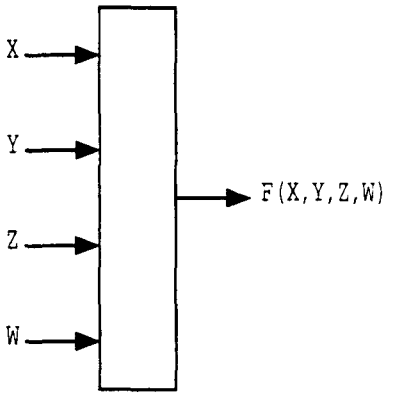


Figure 1: Function on Four Variables

1 The Pattern Theory Approach

The Pattern Theory paradigm focuses on two central ideas shown in this section. The first is functions that the investigator wishes to learn, have low decomposed function cardinality. The second is functions with low decomposed function cardinality are learnable with a relatively small number of samples. In this section, we will present some background on function decomposition and how Pattern Theory uses this as a robust way to find patterns.

The Pattern Theory approach attempts to find an expression for the given data in the training set. The function that it does find will exactly match the training data. However, in order for this function to be useful, not only does it have to match the training data but it has to extrapolate on other data with some confidence. To find that function, we need a measure to rate the function's proposed ability to extrapolate. That measure is obtained when we decompose the function.

Decomposing a function involves breaking it up into smaller subfunctions. These smaller functions are further broken down until the subfunction will no longer decompose. One can see that for a given function the partition space is exponential in the number of variables. The decomposition space is even larger since there are several unique ways subfunctions can be combined and there are several levels of subfunctions possible. The measure that we use to determine the relative abilities of different function decompositions is one of complexity. It is called decomposed function cardinality.

Decomposed function cardinality or DFC, is calculated by taking each subfunction in the decomposition and adding up their cardinality. The cardinality of an n-variable function is 2^n. We illustrate the measure in the following figures. In Figure 1, we have a function on four variables. The function cardinality is $2^4 = 16$. In Figure 2, we show the same function after it has been decomposed. The DFC of this representation for the original function is $2^2 + 2^2 + 2^2 = 12$. The DFC measures the relative complexity for a function. When we search through the possible decompositions for a function, we choose one with the smallest DFC. This decomposition is our learned concept.

The learned function can be used to classify data that are not in the training set. The decomposed representation of the function is one that exhibits more information than the alternative. For example, the Figure 1 representation is essentially a lookup table of inputs and outputs. Figure 2, on the other hand, is a function that is not simply a table. The decomposition, for example, could be two simple functions "or'ed" together.

The only domain knowledge used in this approach is implicit in the representation. In our discussion of relating this approach to a database domain, we will use a natural representation but others are possible. Besides this implicit representation, every attempt was made in developing the theory not to incorporate any such knowledge. The goal was to stay as general as possible.

The important point is that a function with a low DFC has been experimentally determined to be learnable with a small number of samples. The second point is that the functions that we are interested in learning, i.e., functions that are highly "patterned," have a low DFC. These statements will be supported in the quantitative section of this paper. FLASH, an acronym for Function Learning And Synthesis Hot-Bed, was developed to

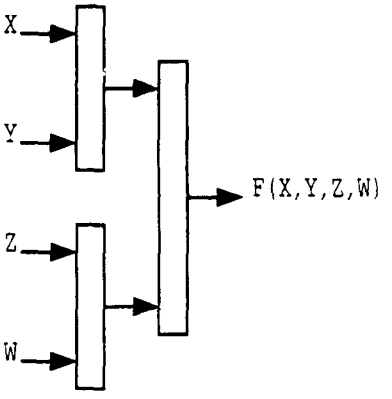


Figure 2: Decomposed Function on Four Variables

explore function decomposition, and pattern finding. The paper will show that the FLASH program exhibits promising results for finding patterns robustly in a domain with a solution useful to the reader. Throughout the paper when we refer to a minimal function decomposition, we use minimal to mean a decomposition such that the DFC is the smallest possible for the entire set of decompositions. It is noted that a given minimal decomposition is not unique. To explore in more rigorous detail about the inner workings of function decomposition or function extrapolation, the reader is referred to [3], [2] and [11].

2 Relating Pattern Theory to KDD

Pattern Theory offers the ability to look for any patterns for a given set of data in a robust way. What is meant by robust is that Pattern Theory has no inherent bias to a particular class of patterns except those of low complexity. This makes Pattern Theory directly applicable to Knowledge Discovery. In general, the approach is to look for some decomposition in a number of binary variables. We can think of a database as a set of m records with n binary fields each. It is noted to the reader that we are simplifying the problem by only allowing binary valued fields. However, since this approach has been proven to be valid for k outcomes instead of just two, we will continue with this model without loss of generality [11]. In fact, our method generalizes to continuous variables as well [10]. We now have a representation that covers all possibilities of data entries. Next, we can take some subset of the records and classify each as a positive or negative example of the proposed concept to be learned. It is also possible to choose a field as the output and attempt to find a description for it in terms of the other fields as in data mining. Our approach will then attempt to find any pattern, if one exists, via function decomposition.

The decomposability of this concept gives a relative measure of "usefulness" to the investigator. If we achieve a minimum decomposition, we have in a sense, extracted a best concept possible with the given information. If this minimum decomposition is small in function cardinality, we have a strong pattern and thus a "useful" concept. If the concept did not decompose, or only decomposed a slight amount, then we can say no "useful" pattern was found or more precisely, there does not exist a strong pattern. It is important to note that we need not have a minimum decomposition in order to exhibit a strong pattern. In fact, if we have a decomposition whose cardinality is (experimentally) small, then the pattern is strong. Moreover with a small decomposition, we can still extrapolate with a small number of errors.

Another esoteric point to make is that it may be possible that our minimum decomposition is less "useful" than a small decomposition. This can happen if our minimum decomposition takes advantage of small unintuitive functions whereas a small decomposition may give us a function more complex, but nonetheless, one we have a name for. We illustrate this point in Table 1. The function F(X,Y) is not a function we have a common name for, however, G(X,Y) is an "xor" function which is more intuitive to the user, even though both F and G have the same cardinality. If a larger function decomposed into more smaller subfunctions like F instead of less subfunctions like G, we can have a minimal decomposition be less useful than a decomposition which is not minimal.

Since Knowledge Discovery has the goal of finding useful patterns in a very large potentially noisy database,

Variables		Functions	
X	Y	F(X,Y) = $\overline{X}$Y	G(X,Y) = X xor Y
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Table 1: The More Intuitive Function

Pattern Theory is directly related. It is ideal in the sense that we can use Pattern Theory to extract the useful information in a small sampling that will have very high accuracy. An advantage Pattern Theory has over a neural network is at the end of "training," we have a readable explanation as a combinational network with universal gates.

Pattern Theory is not yet equipped to handle missing values or noise in the data for all fields. At the time of this writing, there were several ideas on how to handle these situations but none were implemented. Pattern Theory does however, posses the ability to handle missing values in the output fields. In this approach, any attribute that is unknown is treated as a "don't know." In particular, Pattern Theory can extrapolate quite well with significant unknown information. Of course as with any system, if there is too much missing information, FLASH will have an incorrect concept. One advantage of Pattern Theory over other methods is its ability to find patterns with missing information. FLASH will find a pattern in the available data if one exists even though it may not generalise to all of the data. However, this is precisely one of the main issues in Knowledge Discovery.

As well as missing output values, Pattern Theory is also able to handle irrelevant [1] fields. An irrelevant field is found as a vacuous variable in a function decomposition. It is found quite easily. For example, if the concept to be learned in an eight variable function is NOT (X_4), FLASH will find the concept exactly with only 7 samples thus correctly identifying variables $X_1, X_2, X_3, X_5, X_6, X_7$, and X_8 as irrelevant. Missing fields, however, is another story. This is an inherent fault in the database and not the approach. If it turns out that a pattern is found or not found incorrectly because of a missing field, the method to do so simply acted on the available information. Caution needs to be taken when the training information contains inconsistencies. For example, if we have an inconsistency, it would signal a missing field or noisy data and often, we would not be able to determine which error it is. One might argue several inconsistencies would signal a missing field whereas a few would indicate noise. In any case, the same problem is being explored in Pattern Theory as it is with Knowledge Discovery in Databases.

3 Search Complexity

One of the drawbacks to current methods for learning concepts in large databases is the time required to find the answer. Although it has been shown that it is not possible to efficiently learn an arbitrary Boolean concept [7] [8] [12], one can still consider the feasibility of learning for small sizes. The question here is: How small? If we continue with our example of m records with n binary fields each, we can explore the time it would take to learn an arbitrary concept.

Pattern Theory is currently equipped to handle functions with no more than a few dozen variables. In general, finding the minimal function decomposition is an exponential problem if we were to search through all of the decomposition space. However, there is hope. In order to perform in real time for database query, i.e., a few minutes, we are limited to about eight variables. Using each variable to indicate the presence of a field, we are limited to eight fields. The number of records, which is equivalent to training set information or samples, is unlimited. In fact, the more records there are, the better we are equipped to learn the concept.

In practice, an eight field database is quite small and as we increase the variables linearly, the time growth is exponential. Ten to 16 variables can be handled in less than an hour and twenty-four variables can be days. A possible solution to this dilemma using Pattern Theory is to find a decomposition that is not necessarily the

minimum decomposition. Thus, some sort of heuristic search through the set of possible decompositions would increase the number of variables we can handle. If such a heuristic search were available then we would have a technique for finding the underlying pattern without sacrificing very much accuracy. This point is illustrated in [11] but the best methods for finding near minimal decompositions are still unknown.

The consequences of choosing a polynomial time algorithm over the exhaustive search is the ever familiar speed versus accuracy dilemma. Although our polynomial time search will provide useful information for highly patterned concepts, it may not be optimal. For practical discovery issues, this does not appear to be unreasonable. If we are searching for some obscure relation among objects, we need to take the time to consider the possibilities. However, if we wish to perform a user assisted search in as fast and accurate way as possible for correlated fields, a polynomial algorithm is a valid and necessary choice.

4 Learning the Concept

The Pattern Theory approach will always try to find the pattern that exactly matches all of the given information in the simplest way possible. If there is contradictory information, we currently have no method to handle the situation. One alternative being considered is to not include the record in the training set. Although we skirt the problem, we are most likely losing some information. However, preliminary results suggest that this is completely the wrong approach [4]. Instead, it is necessary to have some internal method of handling noise that considers all of the data, correct or not.

Pattern Theory is well suited for the problems associated with irrelevant fields. Irrelevant fields or vacuous variables are found by looking at the column multiplicity [11]. If the field is not present in the learned decomposition, then it has nothing to do with the pattern. For example, this includes the case where a field is pregnancy and we are looking for patterns among men only. Although it is true that the concept among men includes the tautology that they are not pregnant, it is irrelevant. The method also does not omit relevant fields even if they may appear on the surface to be irrelevant. For example, looking for knowledge about a particular disease in a database of patients, it would appear that a zip code field should be irrelevant. However, if there is a pattern based on symptoms as well as geographic region, Pattern Theory is equipped to find the entire relationship.

The case of missing values in fields other than the output needs to be explored. Although we have success when there are missing values for the output field, the general missing value issue has not been addressed. We have some ideas on how to handle the situation. One way is to include the record with every possible value for the missing field duplicating all other entries. This introduces new, potentially erroneous information but may prove better than simply excluding the record for training. Again, a working theory is a future direction for Pattern Theory.

When we have a decomposed function with small cardinality, we have learned a useful concept. This concept can be used as a piece of discovered knowledge and as a tool to extrapolate on additional records. In the end, the relationship of learning concepts in Knowledge Discovery and thus finding a useful pattern, corresponds to finding a function decomposition with a low cardinality in Pattern Theory. Moreover, functions with low DFC are learnable with a relatively small number of samples.

5 Quantitative Results

The last statements are powerful ones, but words are cheap. In this section we exhibit some experimental data on eight variable Boolean functions. The goal is to show the performance of the system and to show the strong correlation between low decomposed function cardinality, the metric used to find patterns, and "patterns" themselves. Also, we wish to show that functions with low DFC are learnable with a relatively small number of samples.

The program FLASH, can handle up to 24 binary variables as mentioned previously but again, only eight for the time required in the database domain. FLASH cannot currently handle missing values for input fields but we demonstrate its ability to do so for output fields. Also, FLASH cannot handle conflicting information for input variables in the training set. We will test unknown outputs by training on known outputs and extrapolating to

	Original Function
F_1	$= (x_1 x_3) + \overline{x}_2$
F_2	$= (x_1 \overline{x}_2 x_3)(x_4 + \overline{x}_6)$
F_3	$= (\overline{x_1 + x_2}) + (\overline{x}_1 x_4 x_6)$
F_4	$= \overline{x}_4$
F_5	$= (x_1 x_2 \overline{x}_4) + (x_3 \overline{x}_5 x_7 x_8) + (x_1 x_2 x_5 x_6 x_8) + (\overline{x}_3 \overline{x}_5)$
F_6	$= x_2 + x_4 + x_6 + x_8$
F_7	$= (x_1 x_2) + (x_3 x_4) + (x_5 x_6) + (x_7 x_8)$
F_8	$= (x_1 \overline{x}_2)$ XOR $(x_1 x_5)$
F_9	$= (x_2$ XOR $x_4)(\overline{x}_1$ XOR $(x_5 x_7 x_8))$
F_{10}	$= (x_1 \Rightarrow x_4)$ XOR $(\overline{x_7 x_8}(x_2 + x_3))$

Table 2: The Functions Tested

others. With these restrictions set aside, we examine FLASH's ability to find patterns in eight binary variables that may have irrelevant fields and missing information in the output field. We also demonstrate FLASH's ability to extrapolate on learned concepts.

Table 2 is a list of all of the functions tested. Table 3 shows the functions with their corresponding minimal DFC. The functions were chosen as an attempt to represent a concept in a database. Functions such as parity or multiply were not included because they have no meaning in this context. It is noted to the reader that many other kinds of "patterned" functions were tested with FLASH besides Boolean Expressions. Some of them include symmetric functions, numerical functions, image functions, and string functions. For results with those types of functions, the reader is referred to Ross et al. [11] and [5].

Table 3 shows an abridged version of our test results with FLASH and C4.5 [9]. The options for C4.5 essentially say that there is no noise in the data and choose the best of 10 trees. These options were not simply chosen at random. Instead, after a careful testing of more than 15 different settings, the options that yielded C4.5's best performance on our benchmark functions was used for comparison [5]. It is noted that we can always increase the number of trees built in C4.5. However, this setting not only made the experiments such that each mechanism had approximately the same amount of CPU time to solve the problem, when we tested 100 trees, the performance increase was not statistically significant.

The tests on the individual functions were as follows. First, each method was given a random set of test cases to train on ranging from 25 to 250 out of a total of 256 possible cases. Once the method was trained, the entire 256 cases were tested and the number of errors were recorded. This procedure was repeated 10 times to yield a maximum, minimum, and average number of errors for a given sample training size. Again, the sample training sizes ranged from 25 to 250 and data was collected in intervals of 25 yielding a total of 10 points of average error and number of training samples. Thus, the total number of runs for each function was 100 of varying sample size. These points were used to construct a learning curve (number of errors vs. training sample size) for each function using C4.5 and Pattern Theory. None of the learning was incremental. All of the runs were independent.

For brevity, a condensed summary of each of the learning curves was also included. For each function, the average number of errors for the entire run was recorded in Table 3, similar to the area under the curve value. It was then possible to compare FLASH and C4.5 with these averages for a given function (middle two columns). The far right two columns of Table 3 show the number of samples necessary before the learning method obtains a concept such that in all ten separate runs, the number of errors was 0. The value at the bottom of the table is the average over all of the functions. The smaller the number here, the better the performance.

Looking at the relationship between a function expression and its DFC, the simpler functions, or more intuitive patterned functions, have a lower DFC. For example, function 1 is less complicated than function 5 and it is a more intuitive relationship between the variables. Thus, its DFC is appreciably lower. It is noted to the reader

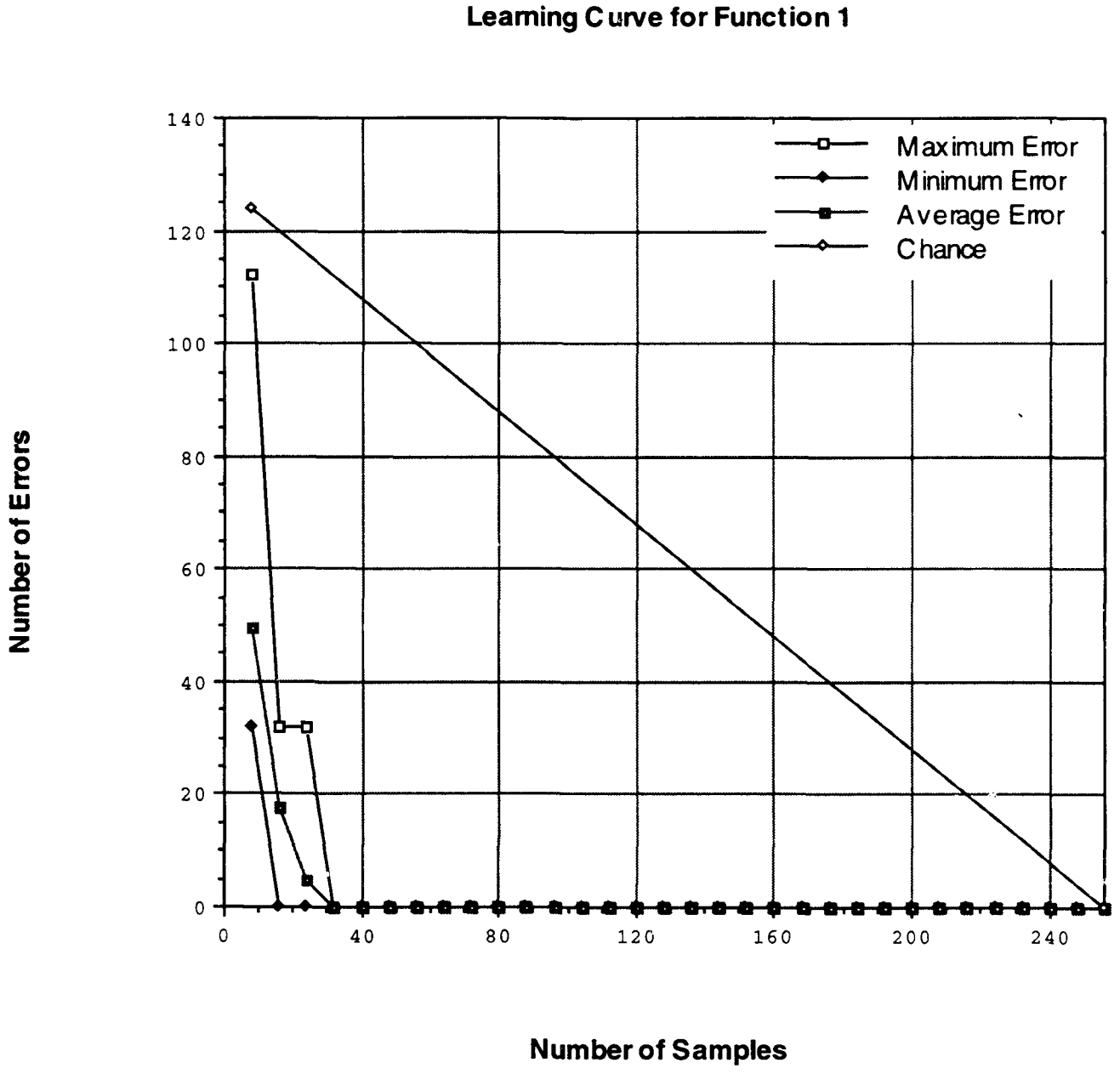


Figure 3: Learning Curve for Function 1

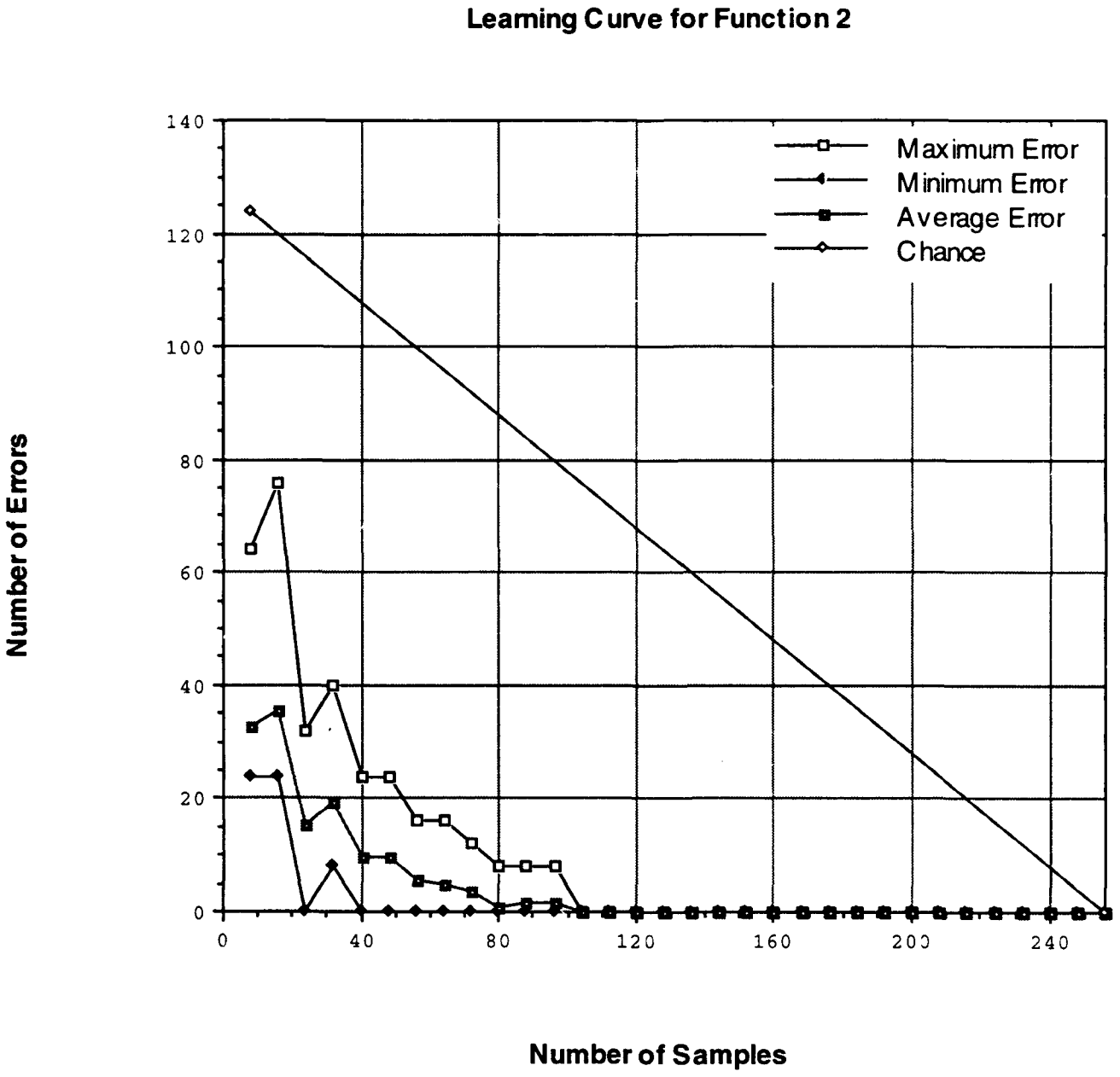


Figure 4: Learning Curve for Function 2

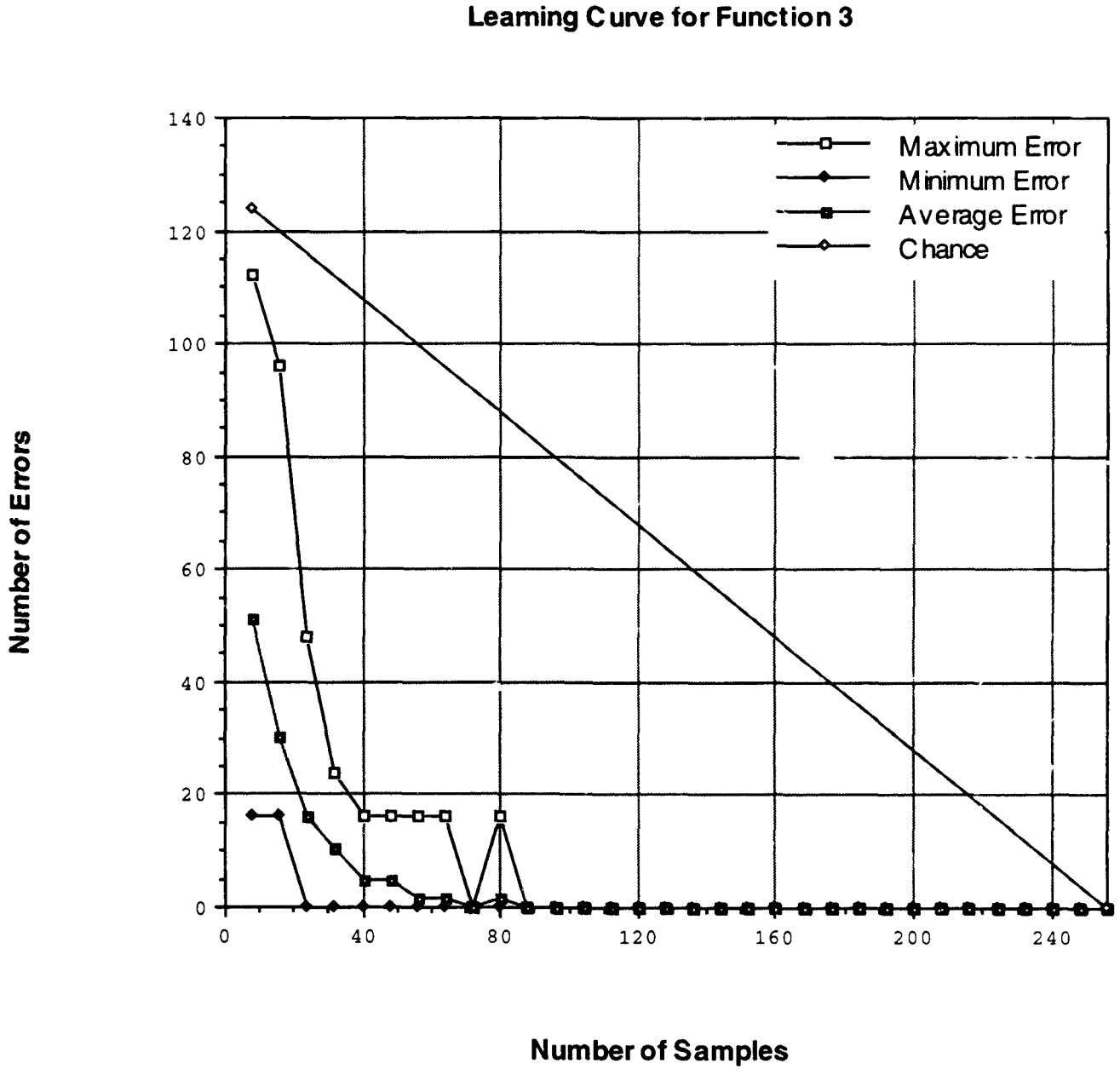


Figure 5: Learning Curve for Function 3

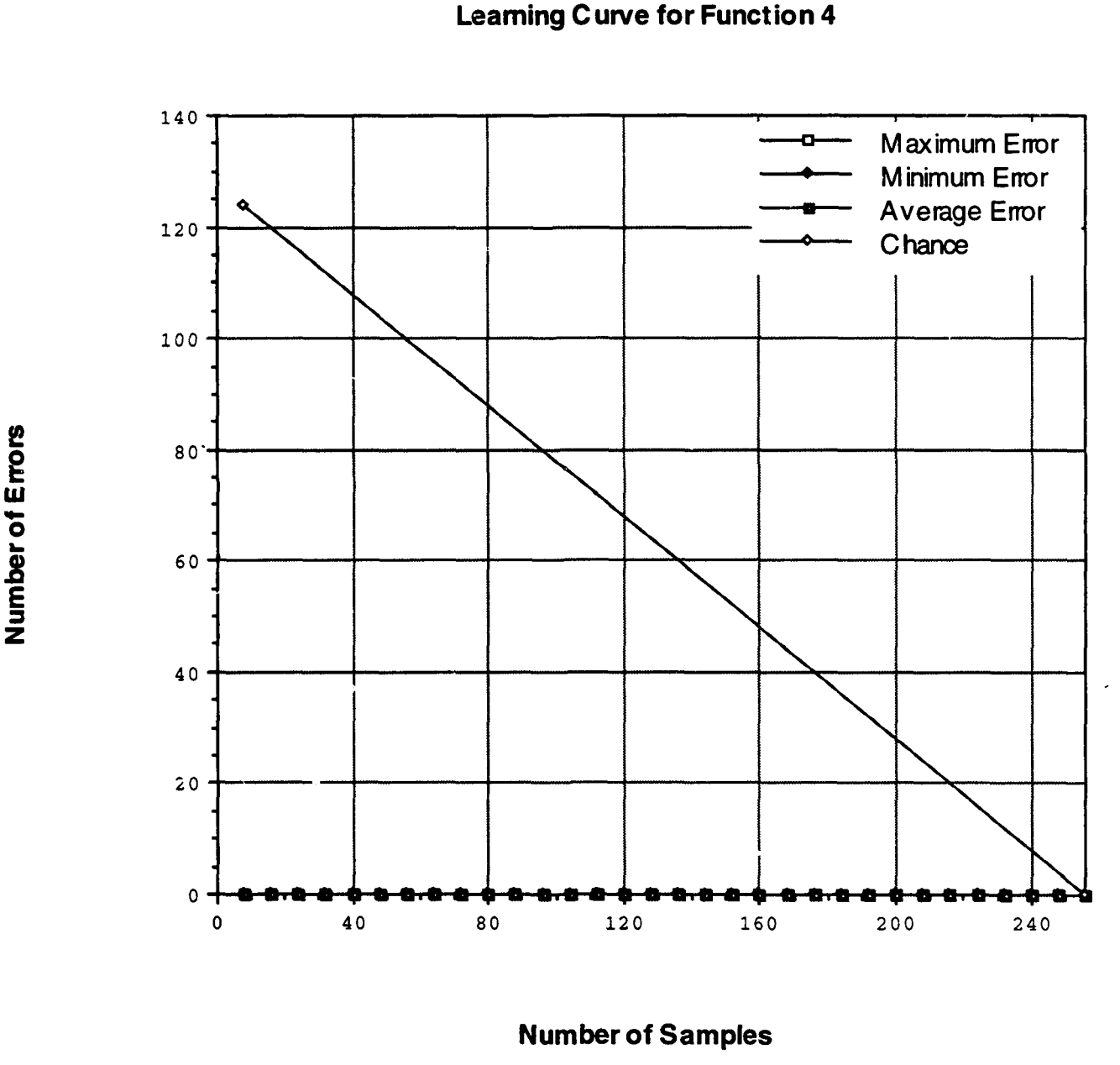


Figure 6: Learning Curve for Function 4

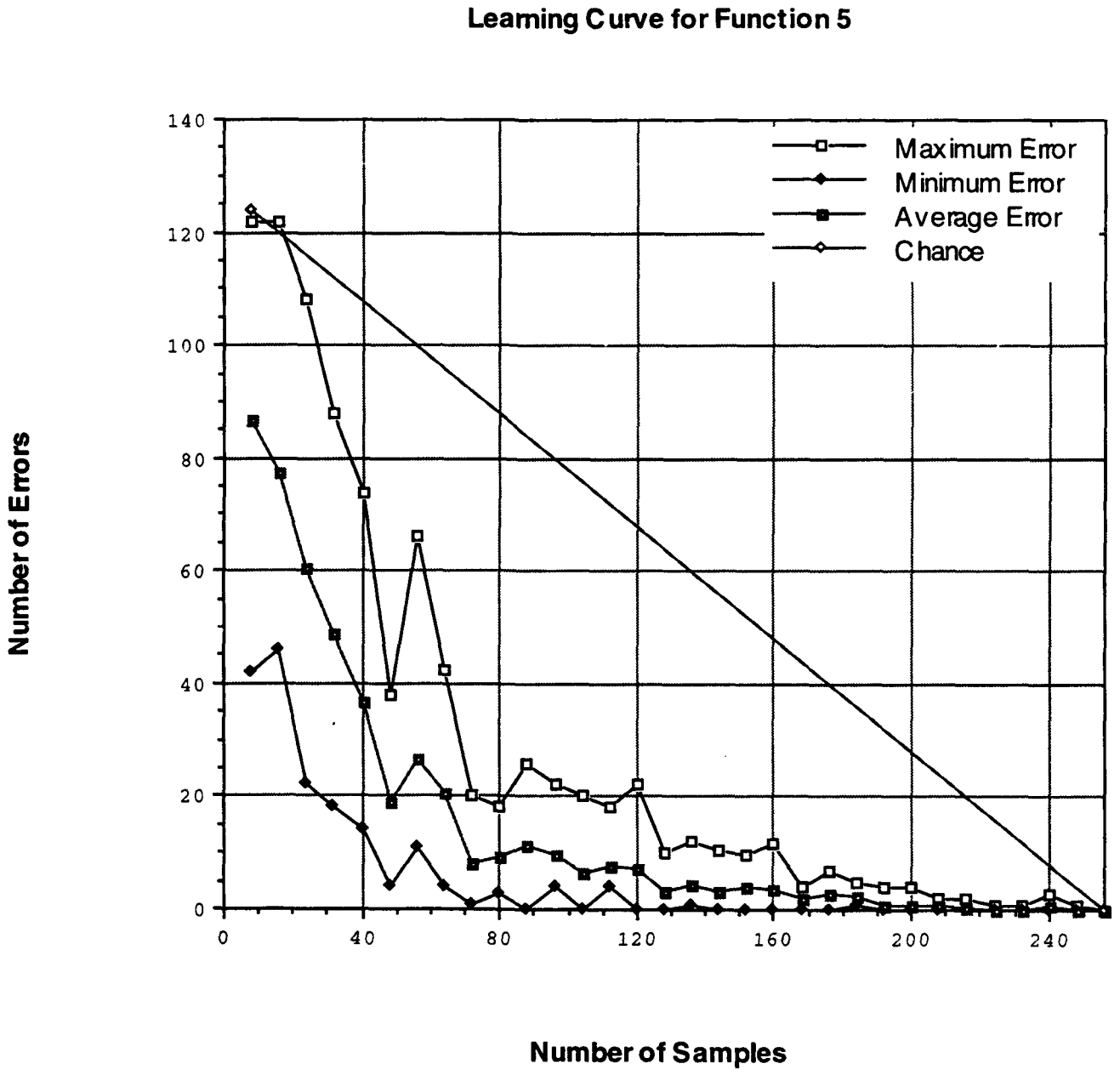


Figure 7: Learning Curve for Function 5

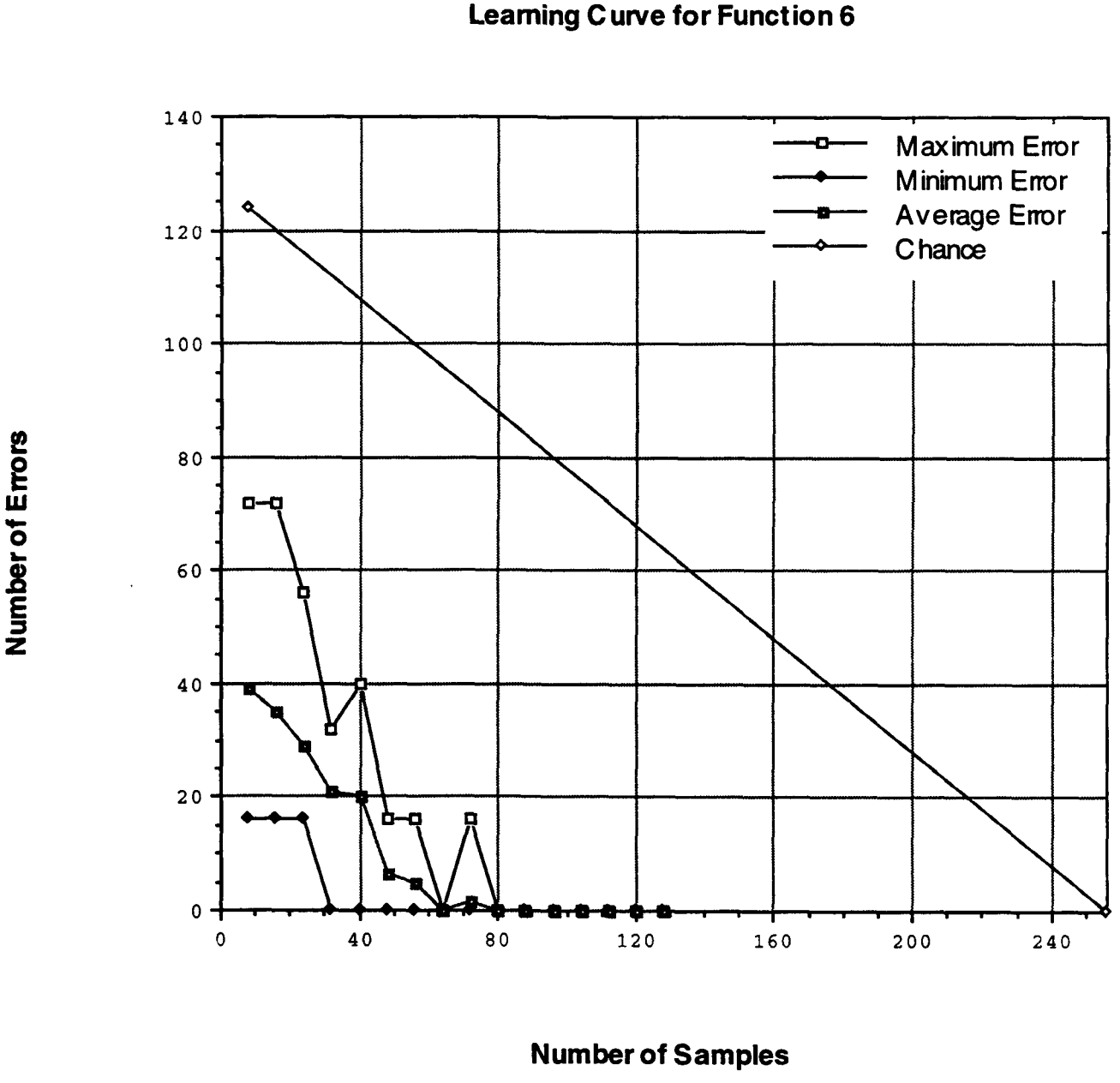


Figure 8: Learning Curve for Function 6

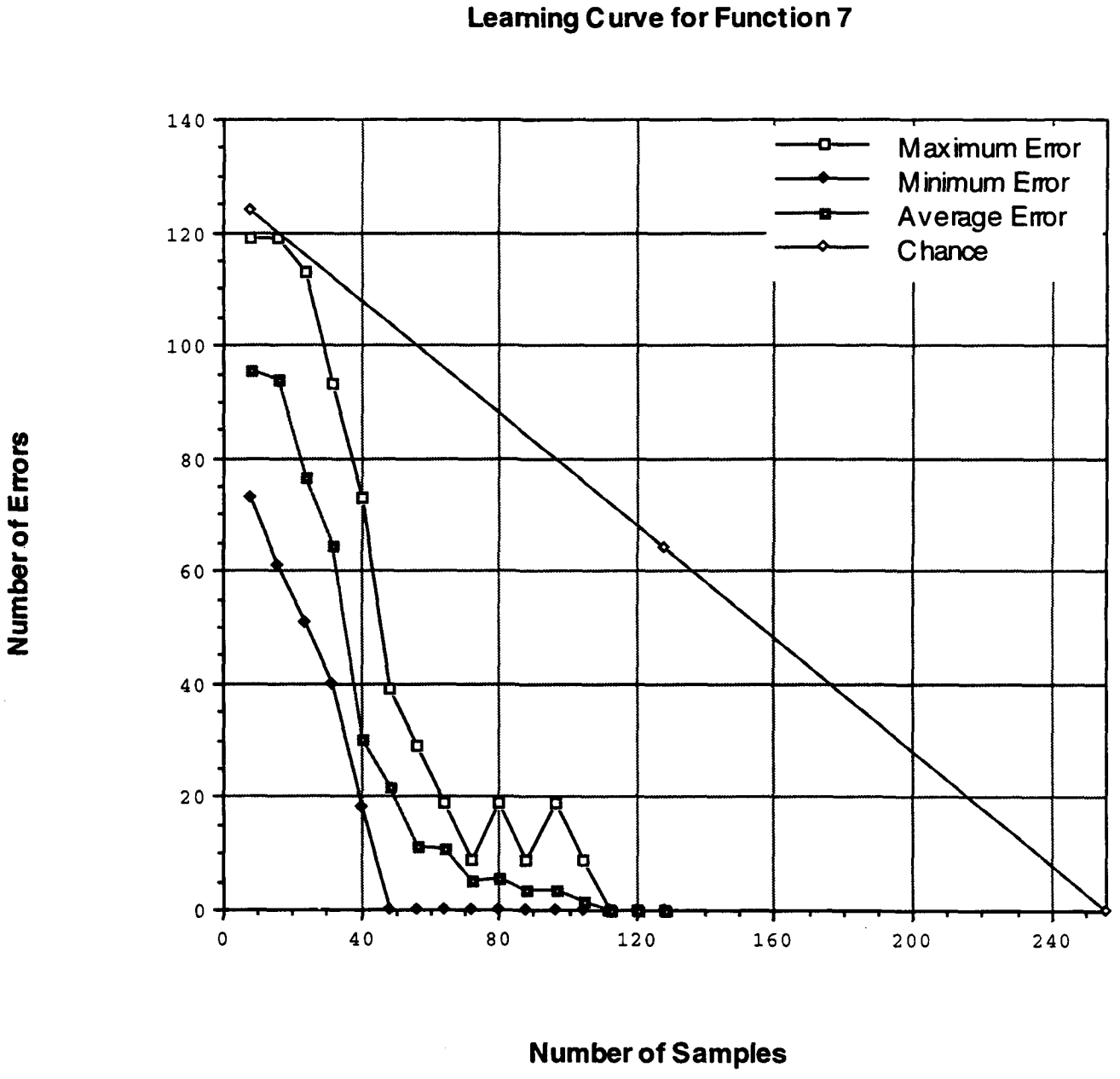


Figure 9: Learning Curve for Function 7

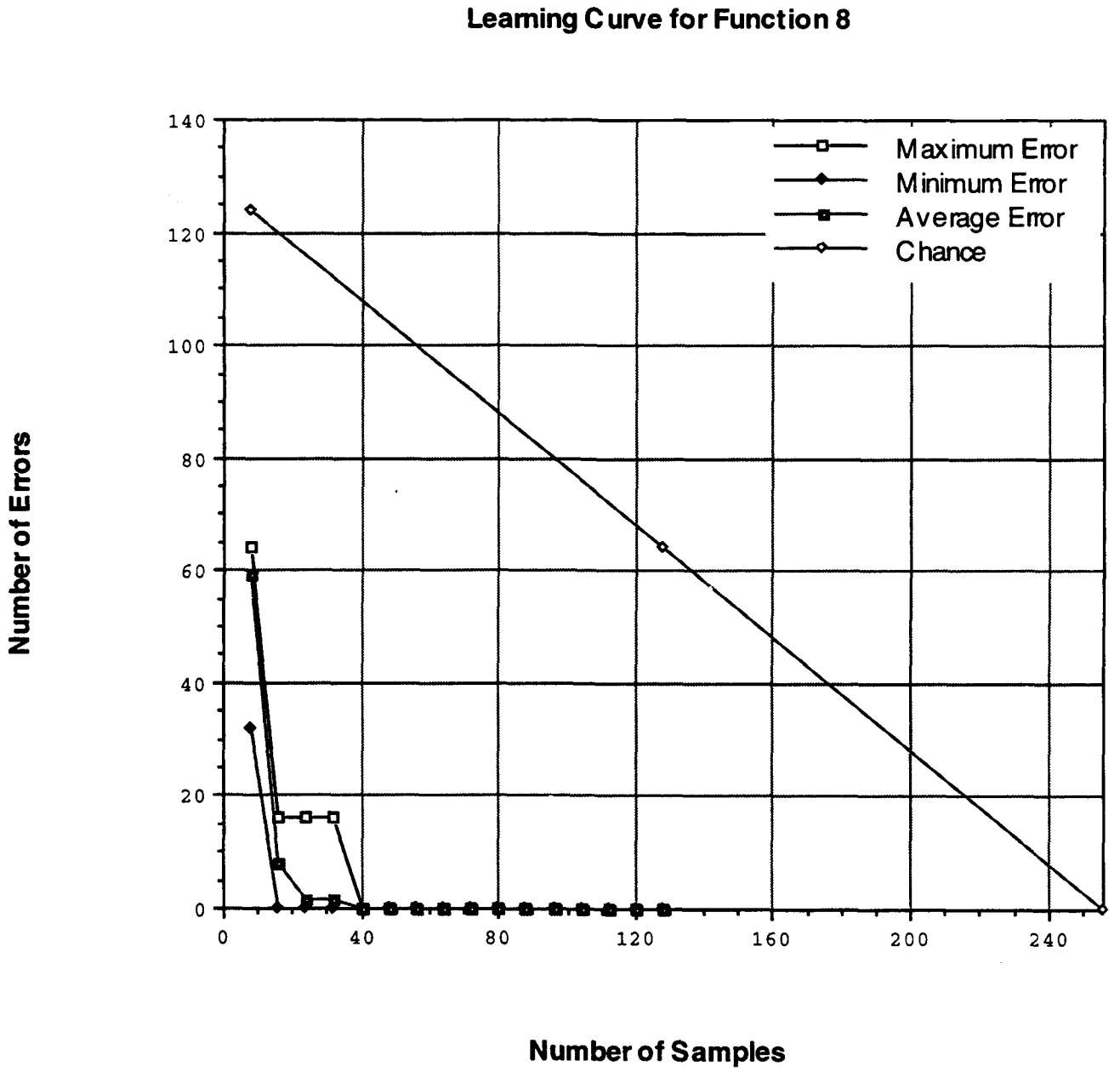


Figure 10: Learning Curve for Function 8

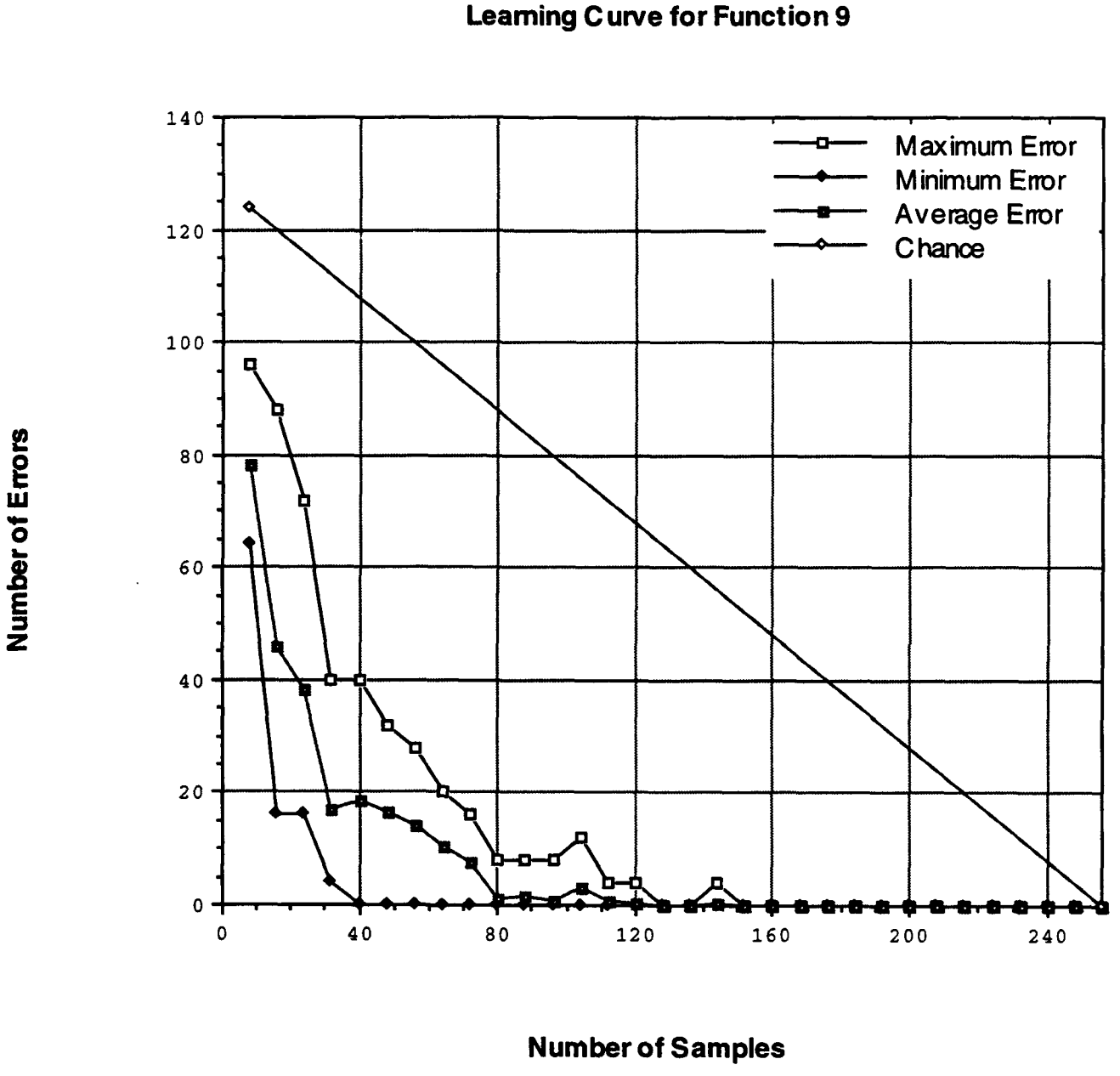


Figure 11: Learning Curve for Function 9

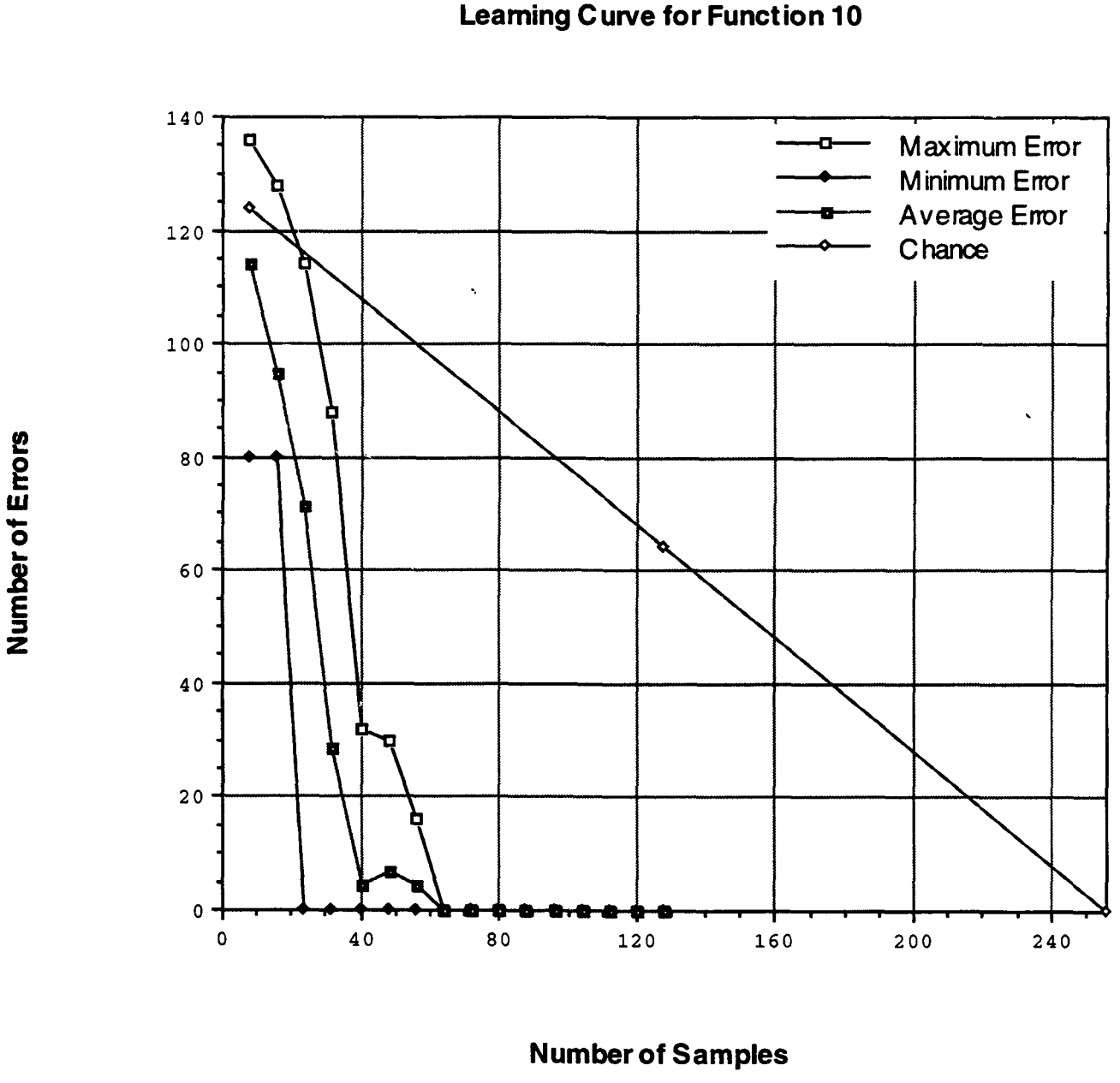


Figure 12: Learning Curve for Function 10

Original Function	Actual DFC	Average Error		# Samples	
		C4.5	Flash	C4.5	Flash
F_4	2	0	0	8	7
F_1	8	0.32	0	31	25
F_8	8	6.35	0	83[1]	25
F_6	12	2.48	3.72	74	67
F_3	12	1.28	2.72	61	76
F_2	16	2.76	2.4	97	126
F_{10}	20	17.52	8.18	200	60
F_9	20	13.79	6.55	224	104
F_7	28	20.69	10.53	256	126
F_5	36	10.52	11.11	249	251
Average		7.57	4.52	128.3[2]	86.7

Table 3: DFC, Mean Error, and # of Samples Needed to Learn

that the DFC range for functions on eight variables is from 0 to 256. Comparing these 10 functions to any eight variable function in general, our examples are relatively simple.

FLASH does a good job at learning all 10 functions. Also, there is a high correlation between the number of samples needed to learn a function and its DFC. In general, the higher the DFC, the more samples that are required to learn the function. The functions with lower DFC require fewer samples to be learned. With the exceptions of functions 5 and 9, FLASH learns all of the functions quickly.

Some of the problems with function 9 may be owed to poor samples. It was unusual to see a difference between function 9 and 10 when they have the same DFC. The reason function 9 was unable to find the true DFC as quickly as function 10 is due to the number of minority elements, vacuous variables, and the decomposition plan itself. All in all, some discrepancy is to be expected but the general trend of DFC measure and number of samples needed to learn, is preserved.

Function 5 is rather involved and is not an intuitive function. Further studies showed that heuristic decomposition searches performed nearly as well in 1/4 the time. We also compared its relative performance against a neural network in which FLASH performed appreciably better. In any case, this function is perhaps too complicated for a typical relationship in a database. The point here is that even if a pattern is not intuitive to us, it can still be discovered by FLASH.

Now that we have shown FLASH's ability to learn and the connection between low DFC, patterned functions, and learnable functions, we wish to examine what concepts FLASH learned. It is important to note that if FLASH has all of the inputs and outputs, it will find a function that exactly matches the information. In all cases but one, FLASH found an equivalent function with considerably less information. Again, FLASH is not restricted in any way as to the representations that it chooses (i.e., there is no bias to the Boolean operators AND, OR, or NOT). The result is that FLASH can find unintuitive functions with the possibility of discovery. For example, function 8 was listed in the Table 2 as (X_1 and $\overline{X_2}$) xor (X_1 and X_5). FLASH found the function: X_1 and $\overline{(X_2 \text{ xor } X_5)}$. This new function is equivalent to the original representation but it is more simplified and in a sense, a discovery. FLASH found functions 4, 6, and 7 exactly as they were represented in the table. Some of the other functions it found were less informative using functions that we have no common name for. However, those unusual representations exactly matched the outputs of the original function.

The point to be made is that the learning method does not bias toward a specific function representation. The trade off is the learned function can be unintuitive to the user. There is room, however, to restrict FLASH to a specific set of functions if the domain calls for it. If for example, there is some database that the only relationships we are interested in includes AND, OR, and NOT, then the concepts we find will only contain those operators

[1] C4.5 with pruning learned this function with 46 samples

[2] The average with the better score for F_8 is 124.6

while still constrained to find the minimum DFC.

One final point to mention about these experiments is the decomposition plan itself. All of the learning curves were generated with the same decomposition strategy. That strategy was to search through all possible partitions at the top level, down to the evaluation of the DFC of each node's grandchild. What this means is for a given breakdown of eight variables into two functions, each was further broken down one additional time. The lowest DFC representations were chosen and recombined. The search was not completely exhaustive. Although the choice of decomposition plans themselves do affect the learning ability of a given function, the functions in this paper were not chosen to make the plan perform better. The choice of plans only affects the amount and order of the search through the decomposition space. Some plans work better on classes of functions because of where they look or where they do not look and not because they somehow take advantage of information about the function.

6 Practical Issues

Thus far, we have explored a different approach to the problem of Knowledge Discovery in Databases. We have showed how Pattern Theory fits into the Knowledge Discovery in Databases ideology. We now want to consider the practical issues in building a real system using the Pattern Theory approach.

Currently, the FLASH program is capable of a database with only eight fields; 16 if we have about an hour, and 24 fields if we can potentially wait for days for our answer. Exploring decompositions that are not mnimal would increase the number of fields we can examine with some loss of the "usefulness" quality that we would like to see in Knowledge Discovery. We are currently exploring methods of pruned searches that still yield near optimal results.

There is also the problem of missing values in input variables. Thus far, we have suggested ways of dealing with the problem but have not explored the consequences quantitatively. A separate but related issue is the problem of conflicting data. Again, we have a good idea of how to try to handle the problem but testing is another focus of future work.

Some theoretical issues that would help formalize the problem include some confidence level or a probably, approximately correct (PAC) model for a "useful concept." At present, we have experimental results that show: If the decomposed function cardinality is less than two times the number of samples, then our confidence level is approximately greater than 50% [11]. Work in this area would give more quantitative relations between confidence of the learned concept and the decomposed function cardinality. Results would not only yield a quantifiable metric for the notion of "pattern," but a confidence measure to boot.

Another issue is the decomposition function cardinality bounds on a polynomial approach to function decomposition. We would like to have a relationship of the polynomial approach to the exhaustive search approach in terms of a bound on the decomposition function cardinality. This bound would allow us to use a polynomial approach with the knowledge of precisely how much confidence and "usefulness" will be sacrificed.

For practicality, we would like to expand FLASH to be able to accept multivalued variables instead of just binary variables . The theoretical results of Ross et al. show that all of the methods generalize. The good news is that changing a two state variable to an n-state variable will not appreciably increase the time required to find the pattern in the fields. The reason is comparisons are made between two numbers, regardless of their value. No additional computation is required. The more important issue is what will happen to the learned concept with this expansion. For example, if one of our fields is temperature over some set of integers, will a concept be difficult to learn if it ranges over a set of temperature values? Also, will the learned concept be "useful?" It may be the case that the decomposed function found is of the form: (L and 60°) or (L and 61°) or ... or (L and 70°). This concept is less "useful" than L and any temperature between 60° and 70° even though we will correctly extrapolate. Exploring different learning curves with n-state variables is a future direction for Pattern Theory.

7 Summary

Although the problems of conflicting data in input or output variables and missing values in input variables still needs development, we have a strong basis for finding patterns in a robust way. Pattern Theory is particularly

well suited for problems that arise dealing with irrelevant fields. The approach simply looks for the pattern within the variables and will not report "obvious" knowledge. We have shown a correlation between low DFC and highly "patterned" functions. We have also shown that only a small number of samples were needed to learn a highly patterned function.

With the theoretical results showing the generalization of function decomposition to n-state variables, the relationship between finding patterns in Pattern Theory and Knowledge Discovery in Databases is clear. The problems and goals are similar. One of the main thrusts for Pattern Theory is to be able to handle hundreds of variables. Analogously, databases are daunting in information and to extract the relevant information, we need to be able to handle large numbers of fields.

References

[1] Hussein Almuallim and Thomas G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 547–552, Anaheim, CA, 1990. AAAI Press.

[2] Robert L. Ashenhurst. The decomposition of switching functions. In *Proceedings of the International Symposium on the Theory of Switching*, April 1957.

[3] Mark L. Axtell, Timothy D. Ross, and Michael J. Noviskey. Pattern theory in algorithm design. In *NAECON Proceedings*. IEEE and AIAA, May 1993.

[4] Jeffrey A. Goldman. Expert systems in preprocessing: A preliminary study of supervised learning with noise using C4.5. Technical Report WL-TR-94-1103, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1994. work in progress.

[5] Jeffrey A. Goldman. Machine learning: A comparative study of pattern theory and C4.5. Technical Report WL-TR-94-1102, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1994. work in progress.

[6] Jeffrey A. Goldman. Pattern theoretic knowledge discovery. In *6th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, November 1994.

[7] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177–221, 1988.

[8] Gregory Piatetsky-Shapiro and William J. Frawley. *Knowledge Discovery in Databases*. AAAI Press, 1991.

[9] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, Palo Alto, California, 1993.

[10] Timothy D. Ross, Jeffrey A. Goldman, David A. Gadd, Michael J. Noviskey, and Mark L. Axtell. On the decomposition of real-valued functions. In *Third International Workshop on Post-Binary ULSI Systems in affiliation with the Twenty-Fourth International Symposium on Multiple-Valued Logic*, 1994.

[11] Timothy D. Ross, Michael J. Noviskey, Timothy N. Taylor, and David A. Gadd. Pattern theory: An engineering paradigm for algorithm design. Final Technical Report WL-TR-91-1060, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1991.

[12] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

8 Appendices

The following appendices were not part of the original paper but were a part of the research work in the area of knowledge discovery in databases. They are included here in order to be complete and for their value to the research group in pattern theory.

There is a short version of this entire work [6] cited here for the reader's convenience.

All of the following data applies to the FLASH program and function decomposition.

A A Detailed Study of Function 5

The first appendix shows 5 graphs all dealing with function 5. For some reason, there was trouble finding this pattern so extra studies were performed in order to gain some type of insight as to the phenomenon.

Graph 1 shows three different decomposition plans and their relative performance. The methods are the same as described in the body of the paper. The decomposition plans are described briefly as follows:

dni0e300 looks at the order of partitions in increasing order and the search is nearly exhaustive.

dnd0v300 looks at the order of partitions in decreasing order and stops at the top level when a vacuous column is found.

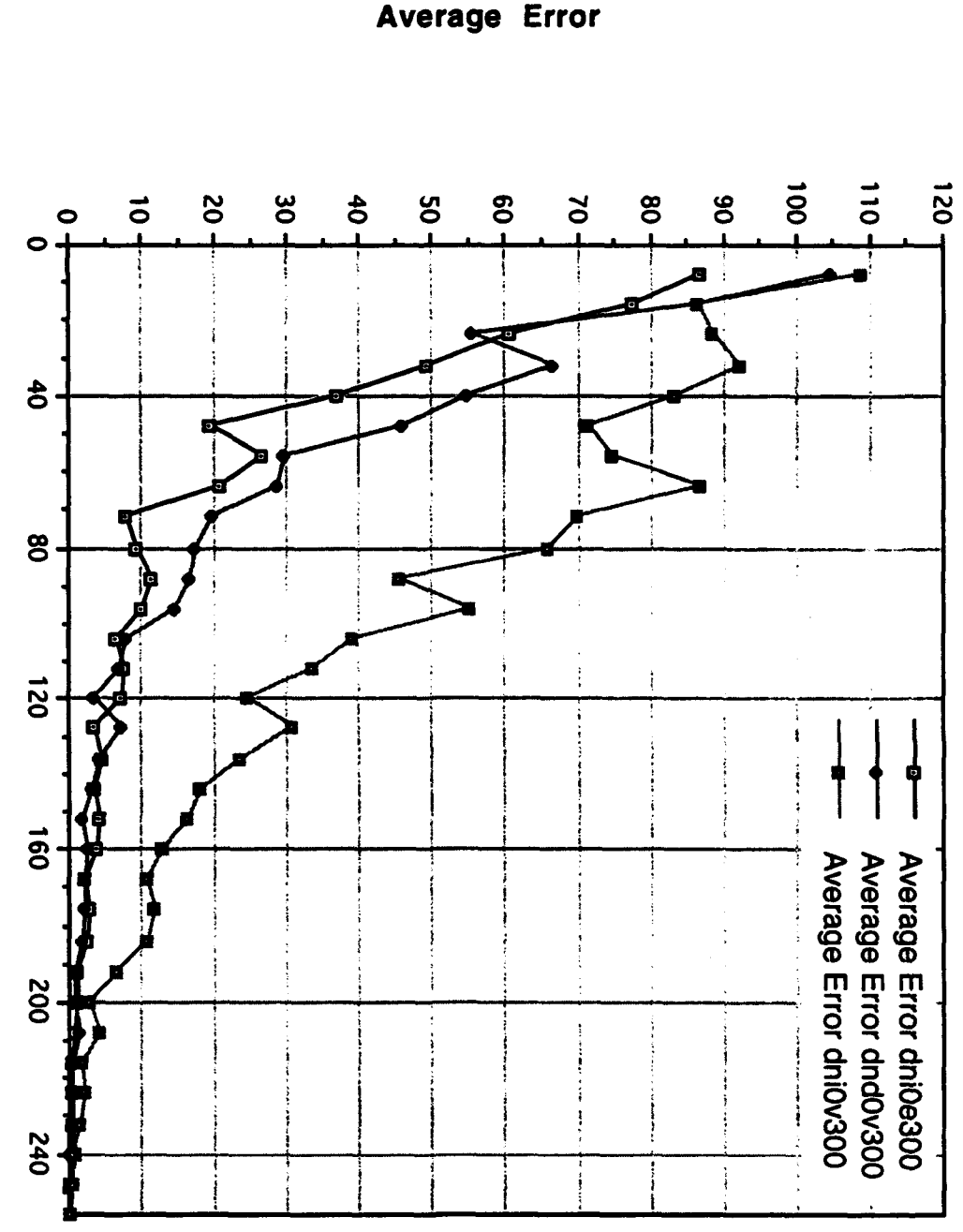
dni0v300 looks at the order of partitions in increasing order and stops searching at the top level when a vacuous column is found.

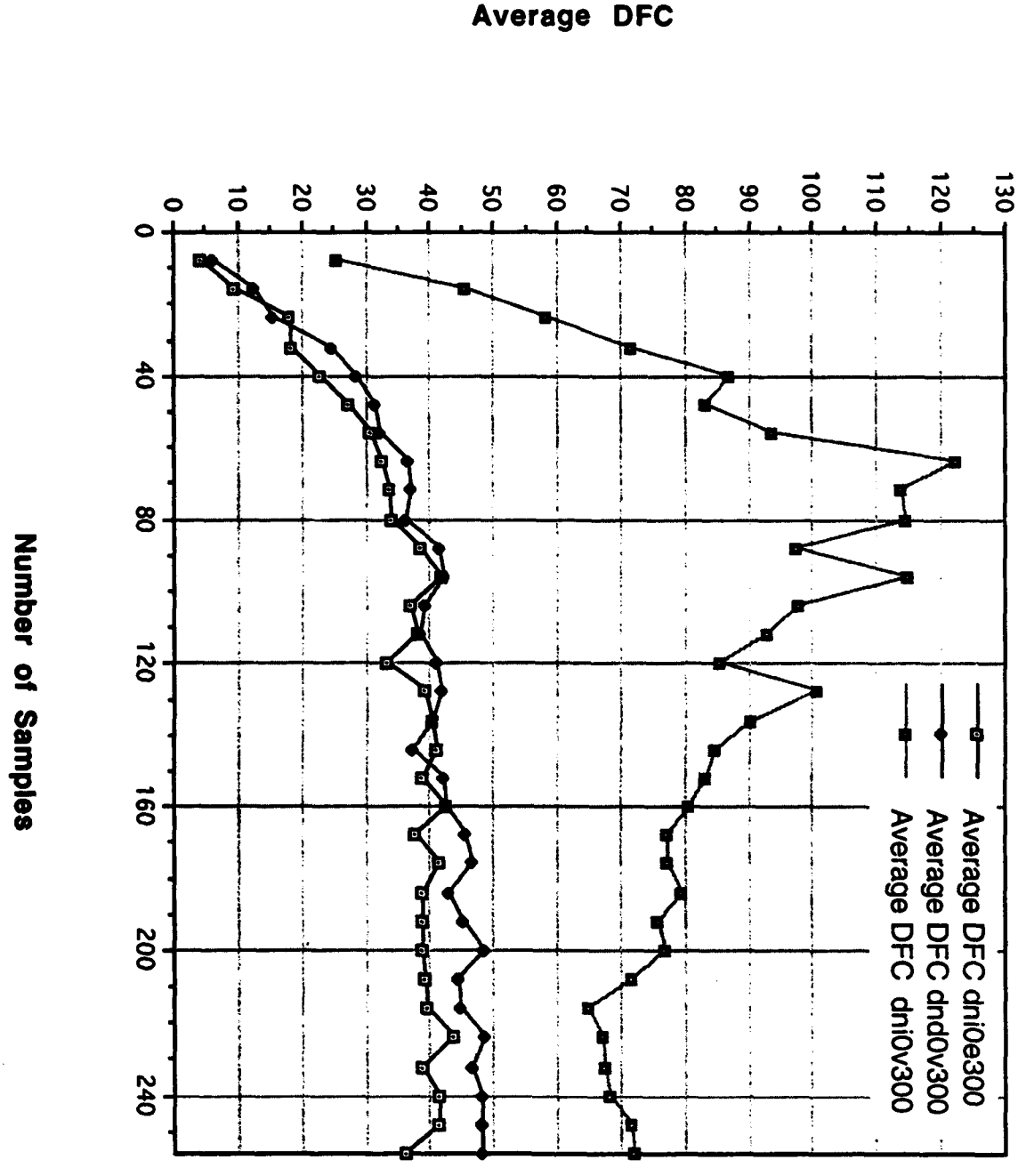
Graph 2 shows the average calculated DFC for each plan as it is given more samples.

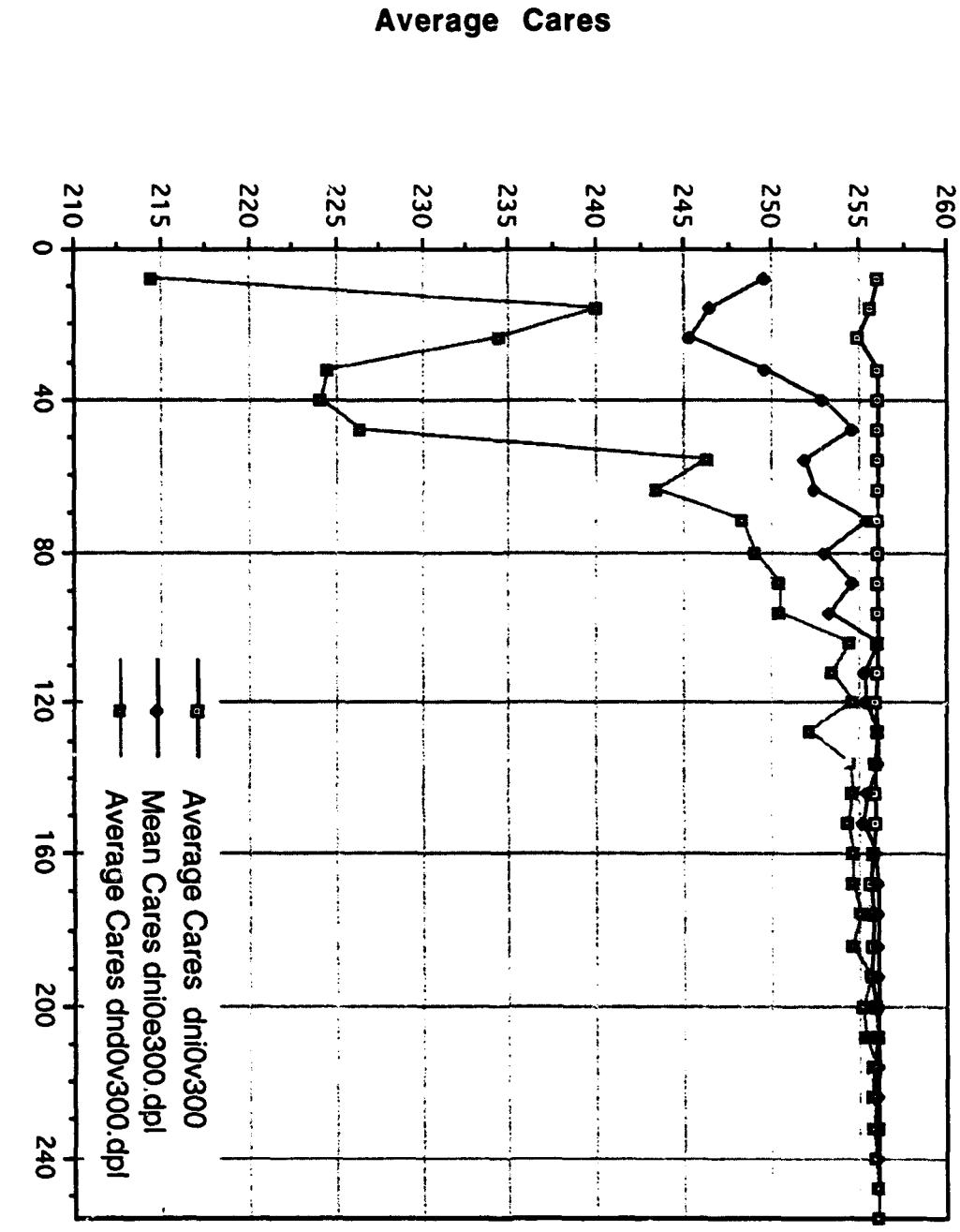
Graph 3 shows the average number of "cares" for each of the decomposition plans.

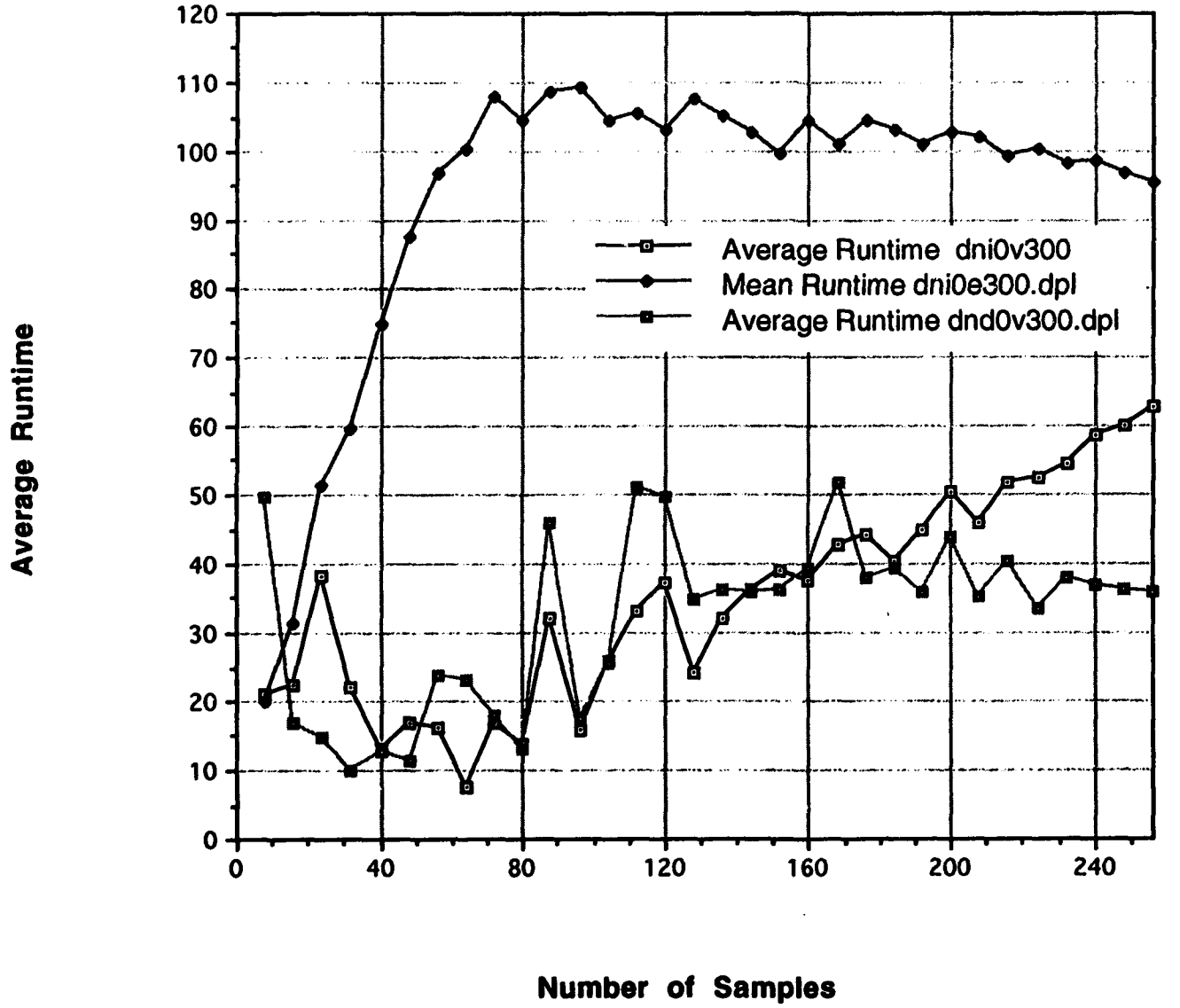
Graph 4 shows the average runtime for each of the decomposition plans.

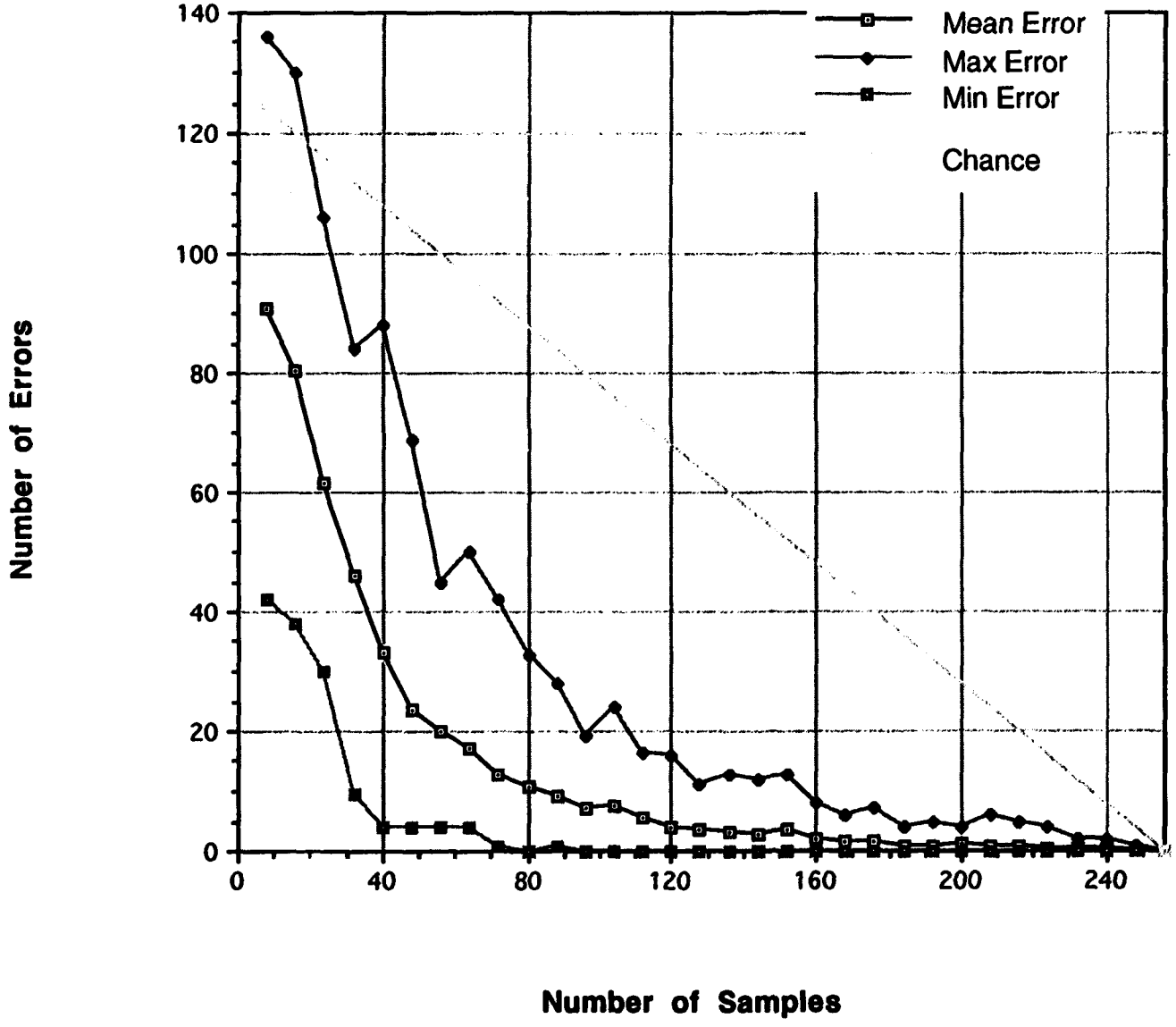
Graph 5 shows the Number of Errors vs. the Number of Samples for various sample sizes as described in the quantitative section of the paper. However, instead of 10 trials per point, there were 100 trials per point.


Average Error
Data from "kdd5three.data"
Number of Samples
Average Error dni0e300
Average Error dnd0v300
Average Error dni0v300
0
10
20
30
40
50
60
70
80
90
100
110
120
0
40
80
120
160
200
240


Average DFC
Data from "kdd5three.data"
Number of Samples
Average DFC dni0e300
Average DFC dnd0v300
Average DFC dni0v300
0
10
20
30
40
50
60
70
80
90
100
110
120
130
0
40
80
120
160
200
240


Average Cares
Data from "kdd5three.data"
Number of Samples
Average Cares dni0v300
Mean Cares dni0e300.dpl
Average Cares dnd0v300.dpl
260
255
250
245
240
235
230
225
220
215
210
0
40
80
120
160
200
240


Data from "kdd5three.data"
Average Runtime
Number of Samples
Average Runtime dni0v300
Mean Runtime dni0e300.dpl
Average Runtime dnd0v300.dpl
0
10
20
30
40
50
60
70
80
90
100
110
120
0
40
80
120
160
200
240


Data from "kdd5c.data"
50 samples per point (dni0e300)
Mean Error
Max Error
Min Error
Chance
Number of Errors
Number of Samples
0
20
40
60
80
100
120
140
0
40
80
120
160
200
240

B The Decomposition Plans

This Appendix lists the decomposition plans themselves as they are entered into the FLASH program.

B.1 DNI0E300

Decomp Plan:
Selection Plan:
0 = use shared variables
12 = method
0 = first part type
0 = stopping condition
Evaluation Plan:
1 = no of partition tests
3 = measure challenger by
3 = measure champ by
4 = threshold in n
1 = champ_multiplier
0 = dp_for_children_is_same
Decomp Plan:
Selection Plan:
0 = use shared variables
20 = method
0 = first part type
1 = stopping condition
30 = stopping condition parameter
Evaluation Plan:
2 = no of partition tests
4 = measure challenger by
1 = measure champ by
4 = threshold in n
1 = champ_multiplier
1 = measure challenger by
1 = measure champ by
4 = threshold in n
1 = champ_multiplier
1 = Random No generator seed (> 0)
0 = dp_for_best_part_children_is_same
Decomp Plan:
Selection Plan:
0 = use shared variables
12 = method
2 = first part type
1 = stopping condition
1 = stopping condition parameter
Evaluation Plan:
1 = no of partition tests
4 = measure challenger by
4 = measure champ by
4 = threshold in n
1 = champ_multiplier
1 = Random No generator seed (> 0)

1 = dp_for_best_part_children_is_same
1 = Random No generator seed (> 0)
1 = dp_for_best_part_children_is_same

B.2 DND0V300

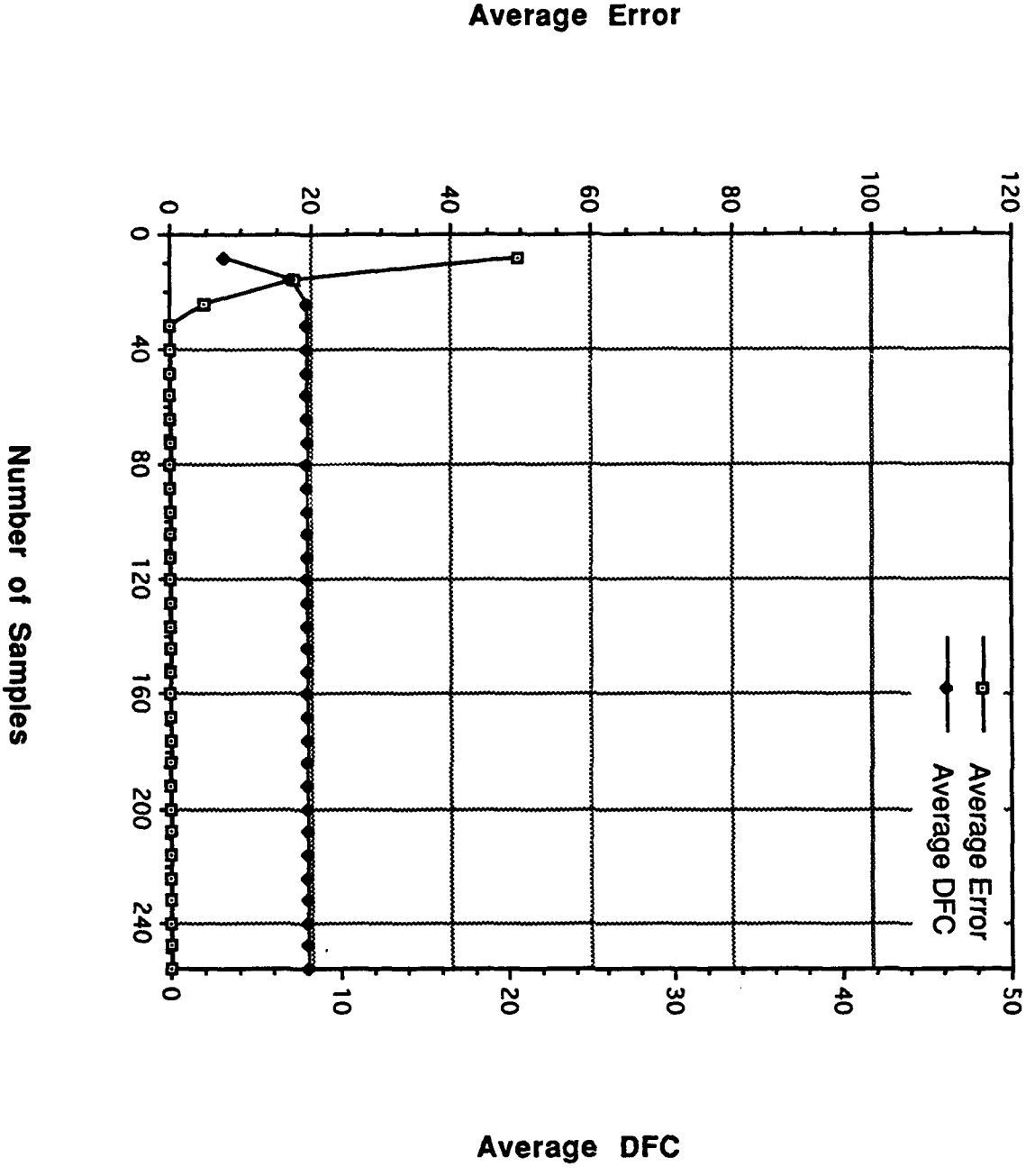
Decomp Plan:
Selection Plan:
0 = use shared variables
11 = method
0 = first part type
4 = stopping condition
Evaluation Plan:
1 = no of partition tests
3 = measure challenger by
3 = measure champ by
4 = threshold in n 1 = champ_multiplier
1 = dp_for_children_is_same
1 = Random No generator seed (> 0)
1 = dp_for_best_part_children_is_same

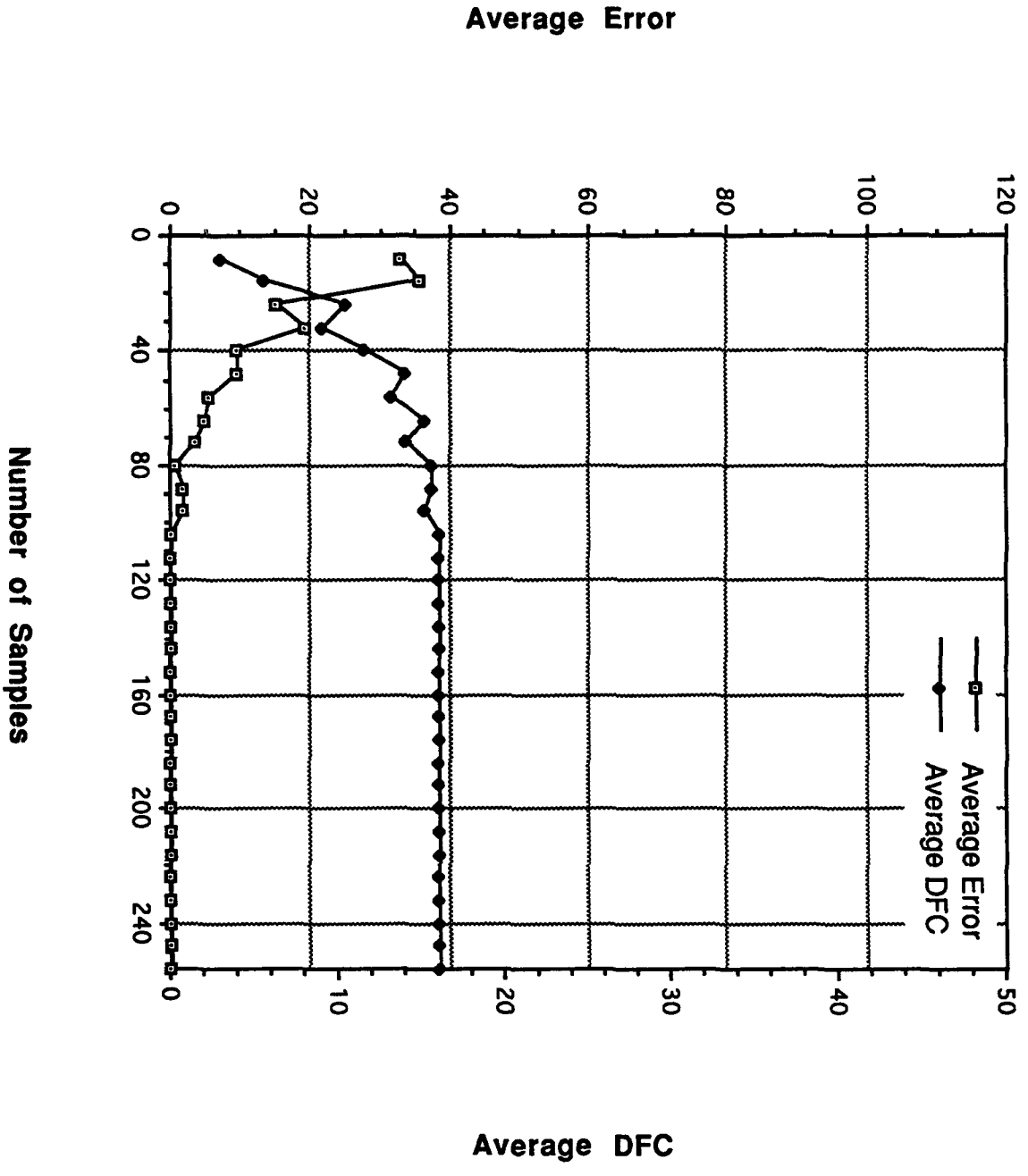
B.3 DNI0V300

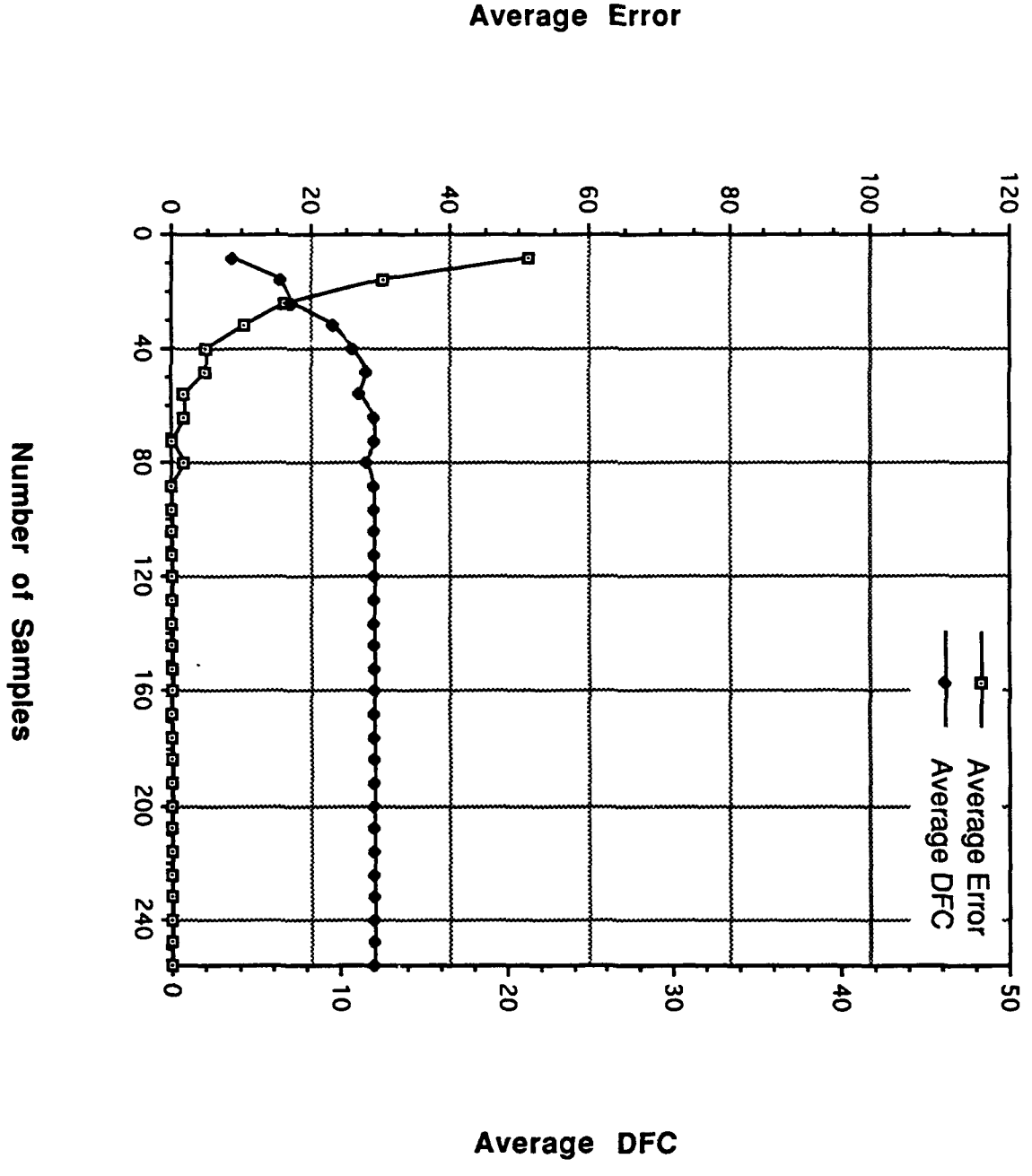
Decomp Plan:
Selection Plan:
0 = use shared variables
12 = method
0 = first part type
4 = stopping condition
Evaluation Plan:
1 = no of partition tests
3 = measure challenger by
3 = measure champ by
4 = threshold in n
1 = champ_multiplier
1 = dp_for_children_is_same
1 = Random No generator seed (> 0)
1 = dp_for_best_part_children_is_same

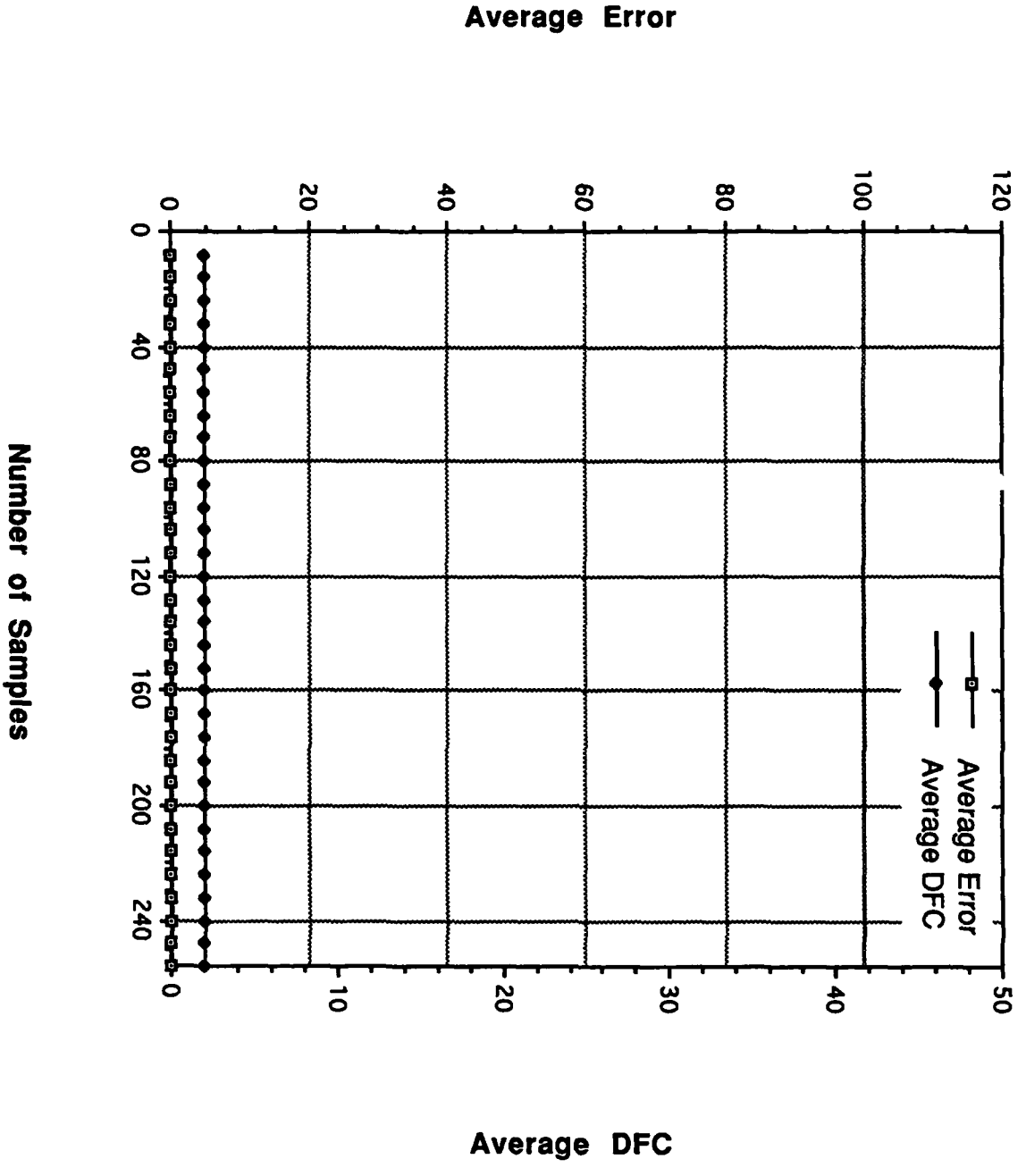
C The Complete Graphs of All Ten Functions (FLASH Only)

This Appendix shows 10 graphs, one for each function listed in Table 2 in the paper body. The graphs show the average error with the average calculated DFC for comparison. Each function is explicitly listed at the top along with the actual DFC, the number of vacuous variables, and the number of minority elements for convenience.

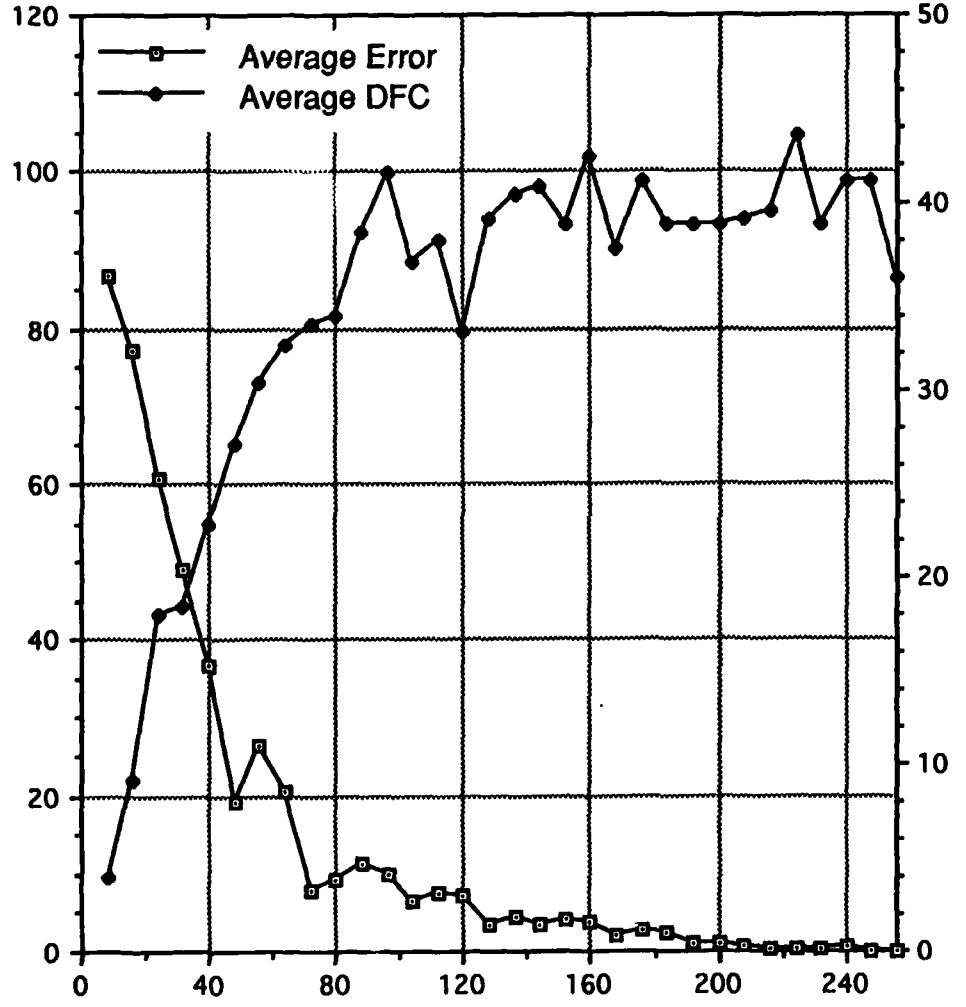

KDD 1 = X1 X3 + X2'
DFC = 8
5 Vacuous Variables 96 Minority Elements
Average Error
Average DFC
Number of Samples
Average Error
Average DFC
0
20
40
60
80
100
120
0
10
20
30
40
50
0
40
80
120
160
200
240

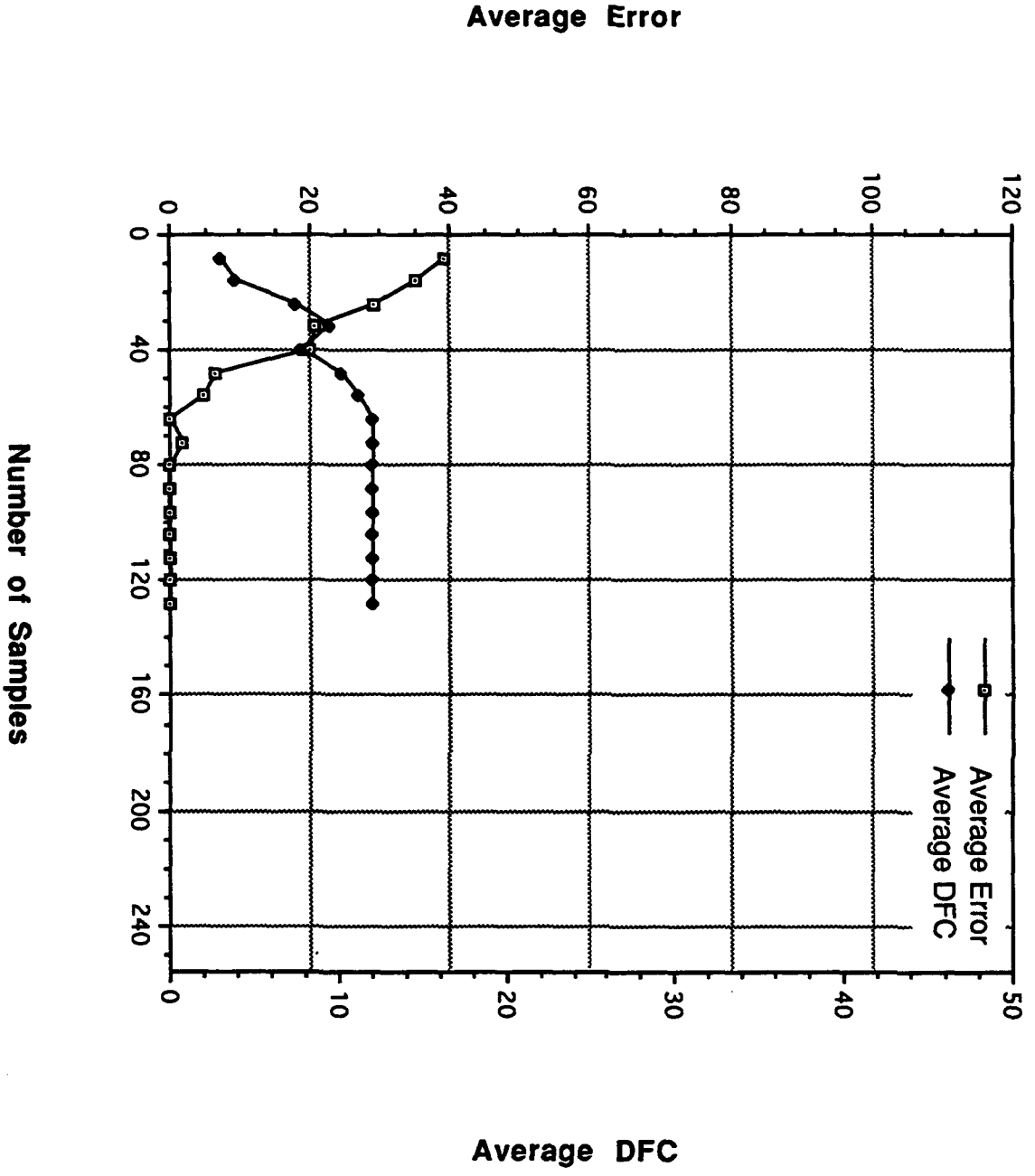

KDD2 = X1 X2' X3 (X4 + X6')
DFC = 16
3 Vacuous Variables 24 Minority Elements
Average Error
Average DFC
Number of Samples
0
20
40
60
80
100
120
0
10
20
30
40
50
0
40
80
120
160
200
240
Average Error
Average DFC

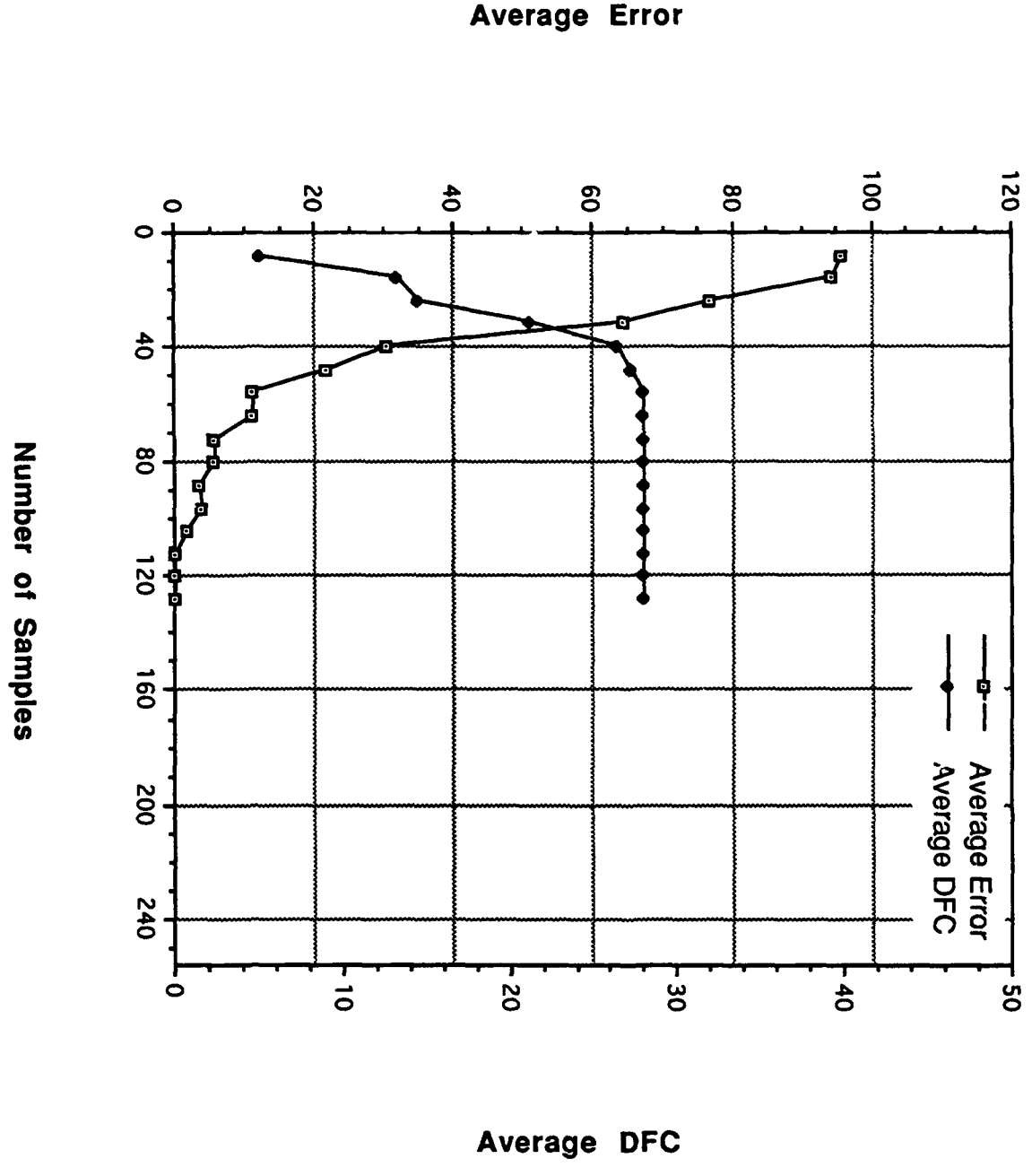

KDD 3 = NOT (X1 + X2) + X1' X4 X6
DFC = 12
4 Vacuous Variables 80 Minority Elements
Average Error
Average DFC
Number of Samples
Average Error
Average DFC
0
20
40
60
80
100
120
0
10
20
30
40
50
0
40
80
120
160
200
240


Average Error
KDD4 = X4'
DFC = 2
7 Vacuous Variables 128 Minority Elements
Average Error
Average DFC
120
100
80
60
40
20
0
50
40
30
20
10
0
0
40
80
120
160
200
240
Average DFC
Number of Samples

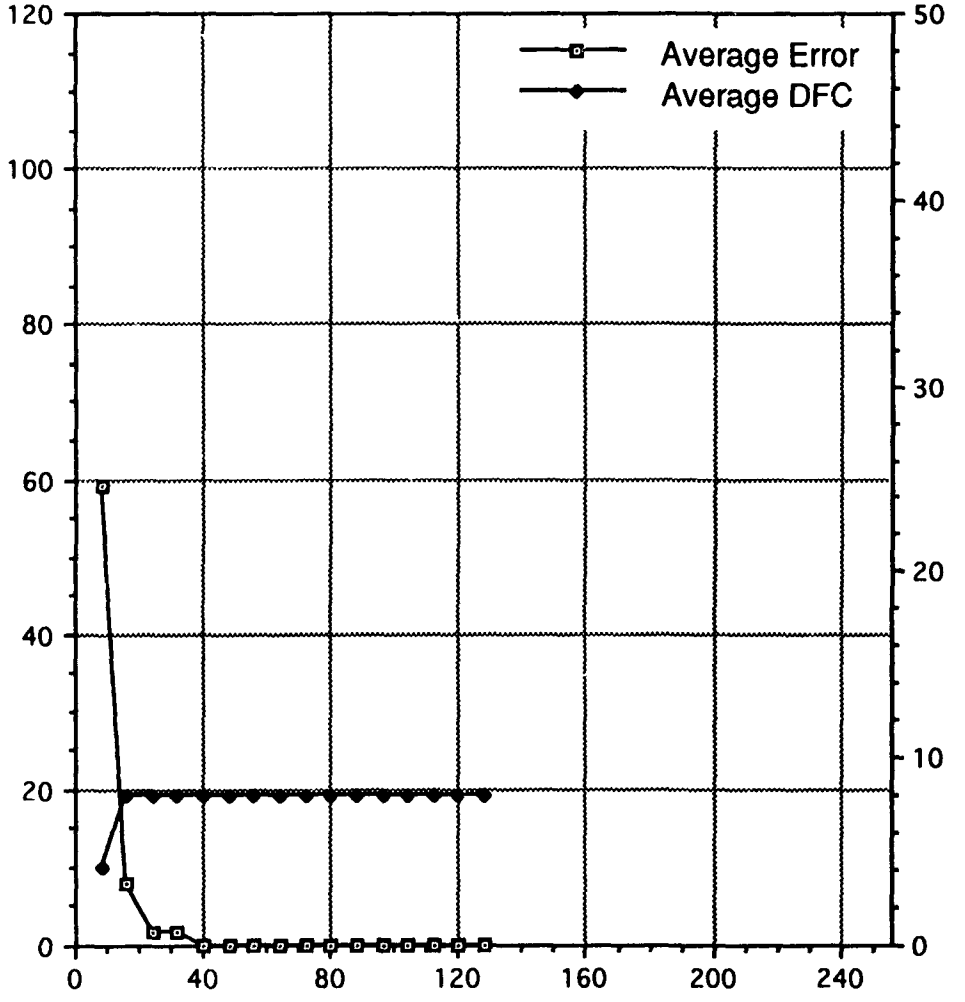
KDD 5 = X1 X2 X4' + X3 X5' X6 X8 +
DFC = 36 (X1 X2 X5 X6 X7 X8) + X3' X5'
0 Vacuous Variables 106 Minority Elements

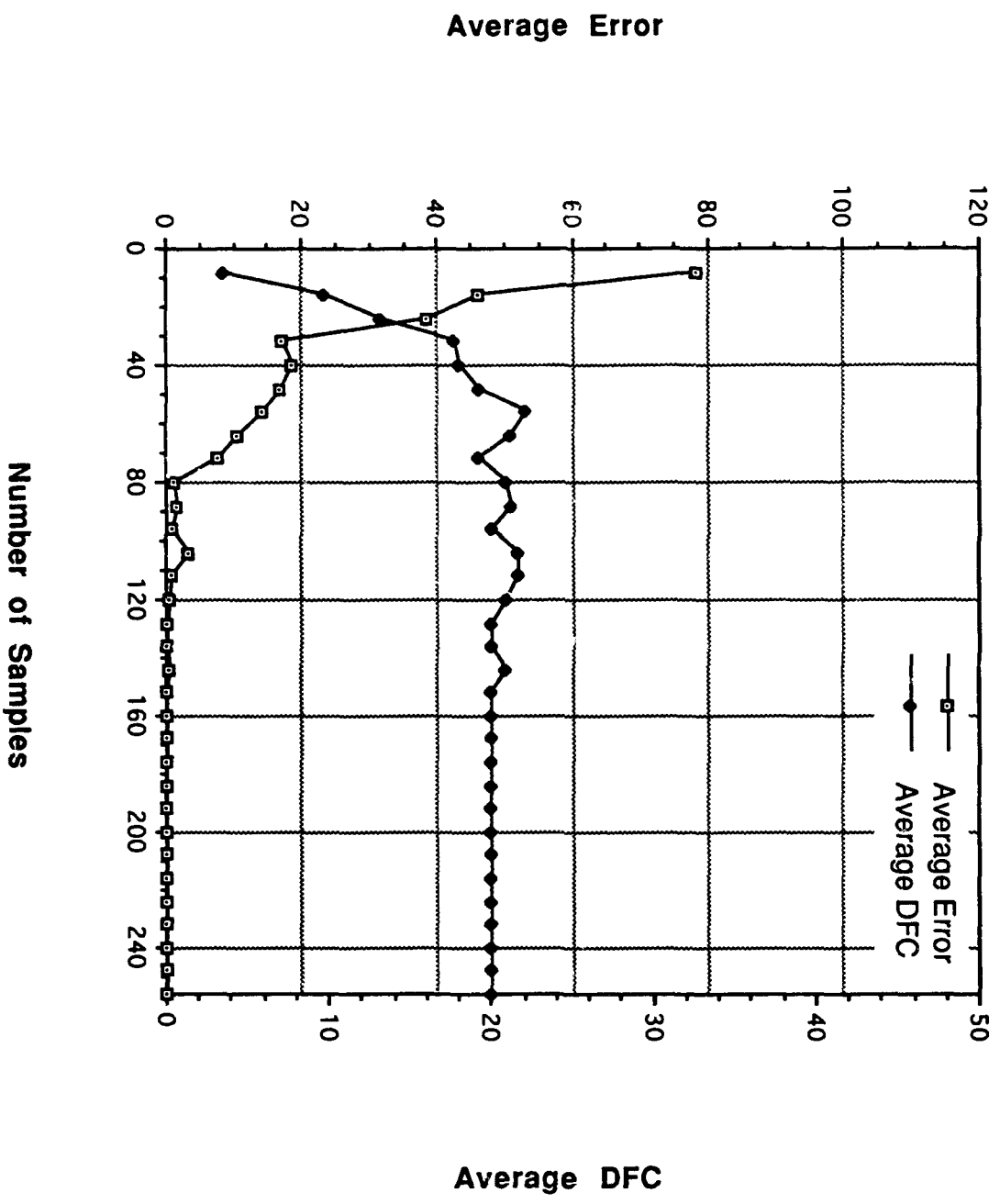


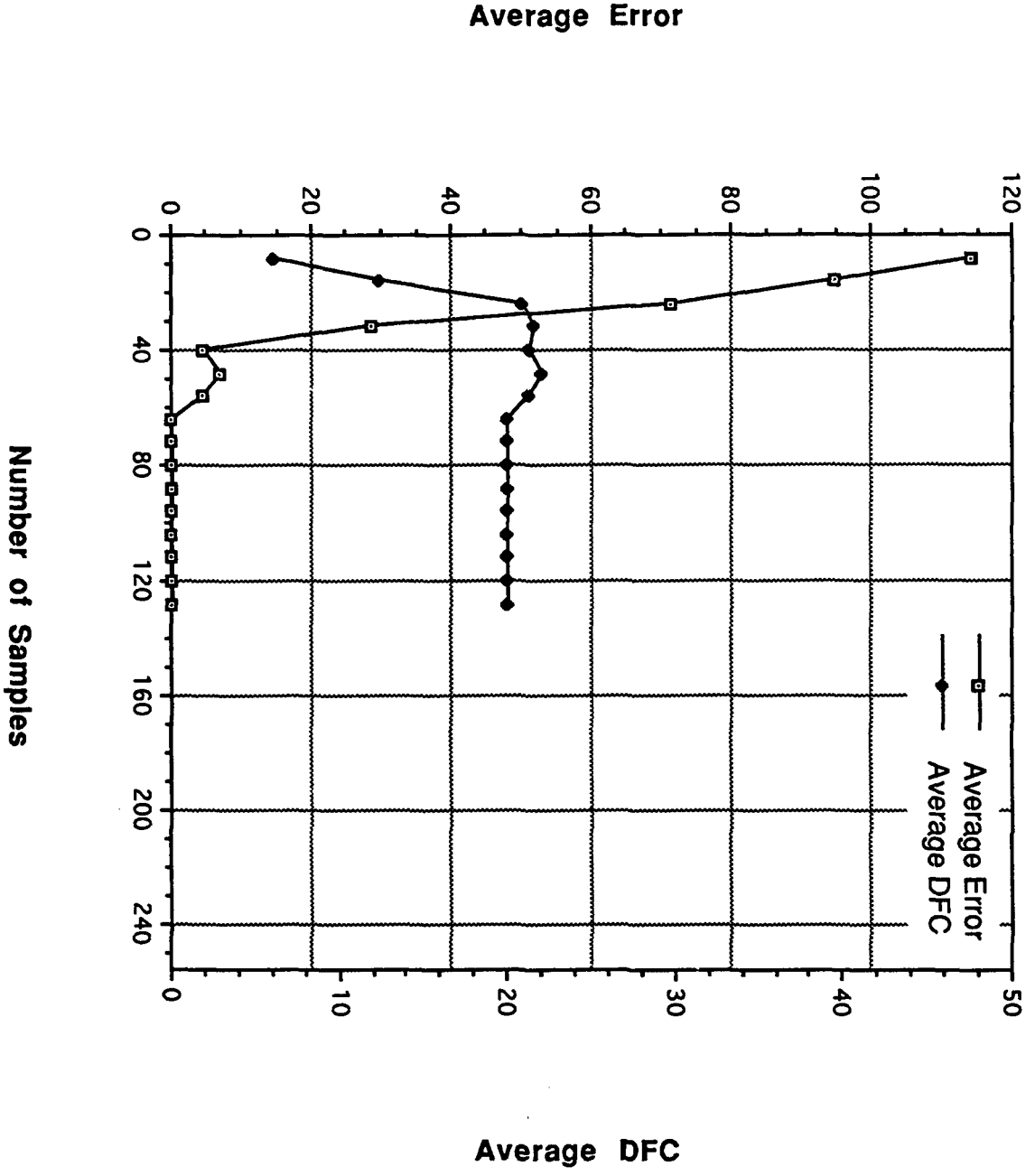

KDD 6 = X2 + X4 + X6 + X8
DFC = 12
4 Vacuous Variables 16 Minority Elements
Average Error
Average DFC
Number of Samples
0
20
40
60
80
100
120
0
40
80
120
160
200
240
0
10
20
30
40
50


KDD 7 = X1X2 + X3X4 + X5X6 +X7X8
DFC = 28
No Vacuous Variables 81 Minority Elements
Average Error
Average DFC
Number of Samples
Average DFC
0
20
40
60
80
100
120
0
10
20
30
40
50
0
40
80
120
160
200
240

KDD 8 = X1X2' XOR X1X5
DFC = 8
5 Vacuous Variables 64 Minority Elements



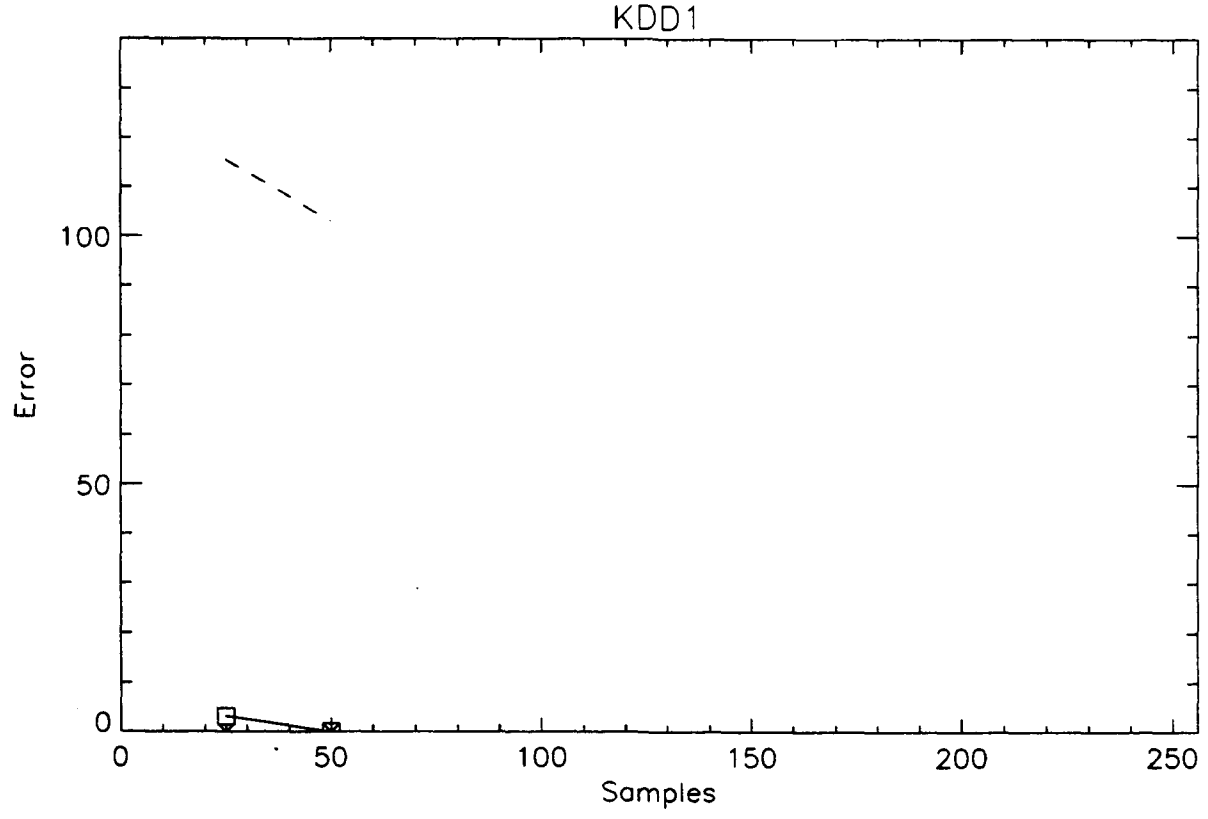

KDD 9 = (X2 XOR X4)(X1' XOR (X5X7X8))
DFC = 20
2 vacuous variables 96 Minority Elements
Average Error
Average DFC
Number of Samples
Average Error
Average DFC
0
20
40
60
80
100
120
0
10
20
30
40
50
0
40
80
120
160
200
240

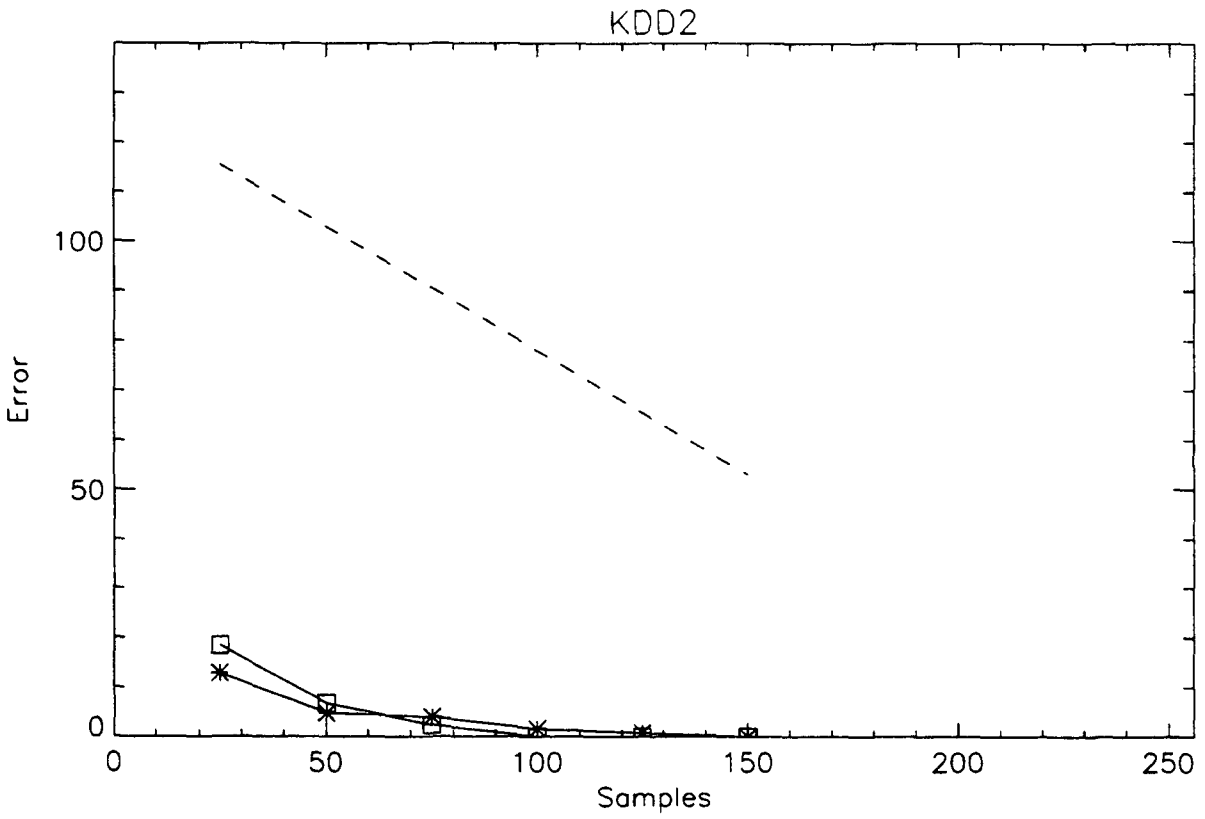

KDD 10 = (X1 => X4) XOR
DFC = 8 (NOT (X7 X8))(X2 + X3)
2 Vacuous Variables 120 Minority Elements
Average Error
Average DFC
Number of Samples
Average Error
Average DFC
0
20
40
60
80
100
120
0
40
80
120
160
200
240
0
10
20
30
40
50

D Comparison Graphs of All Ten Functions (C4.5 and FLASH Together)

This Appendix shows a comparison of the 10 functions. The graphs show the average error for each sample size with both methods. This data was used in [6] in a condensed form. The graphs are shown here explicitly.

As before, each point was an average over 10 separate runs with a given sample size.

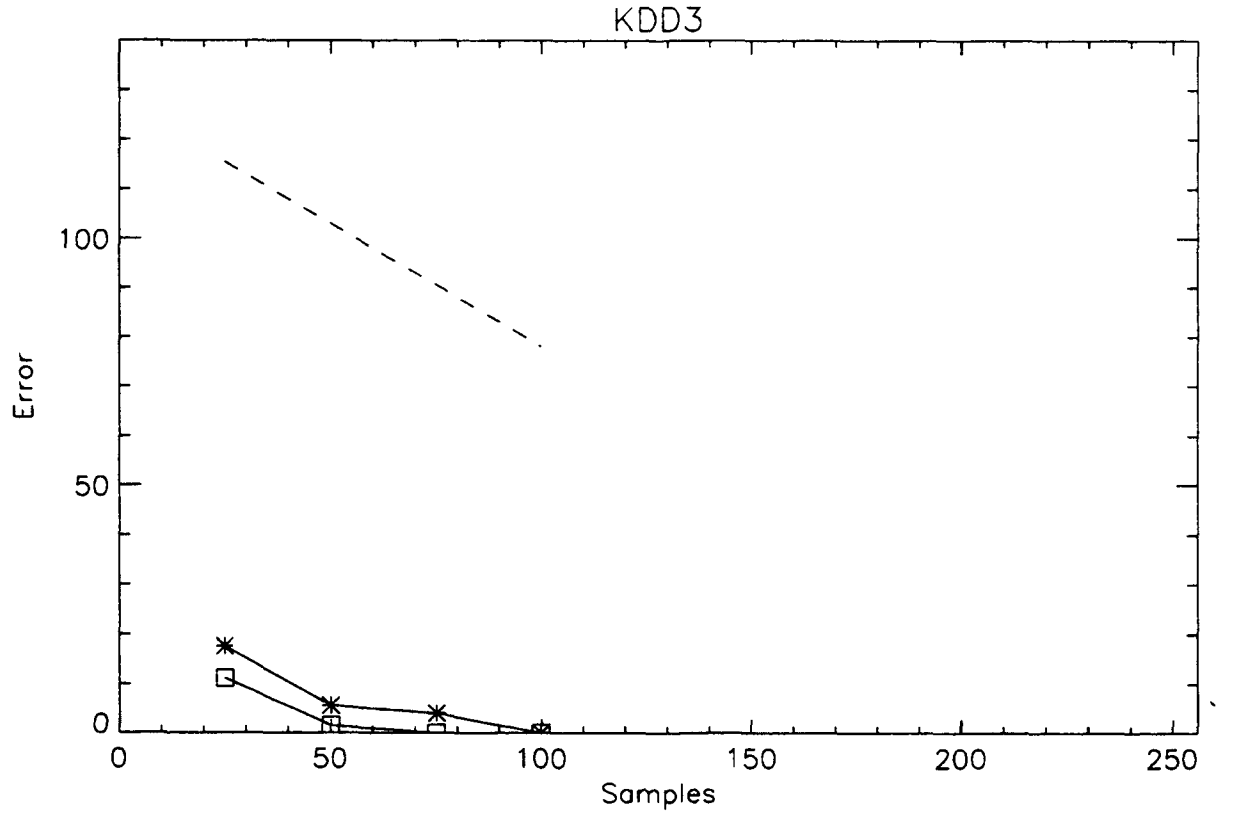

KDD1
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10

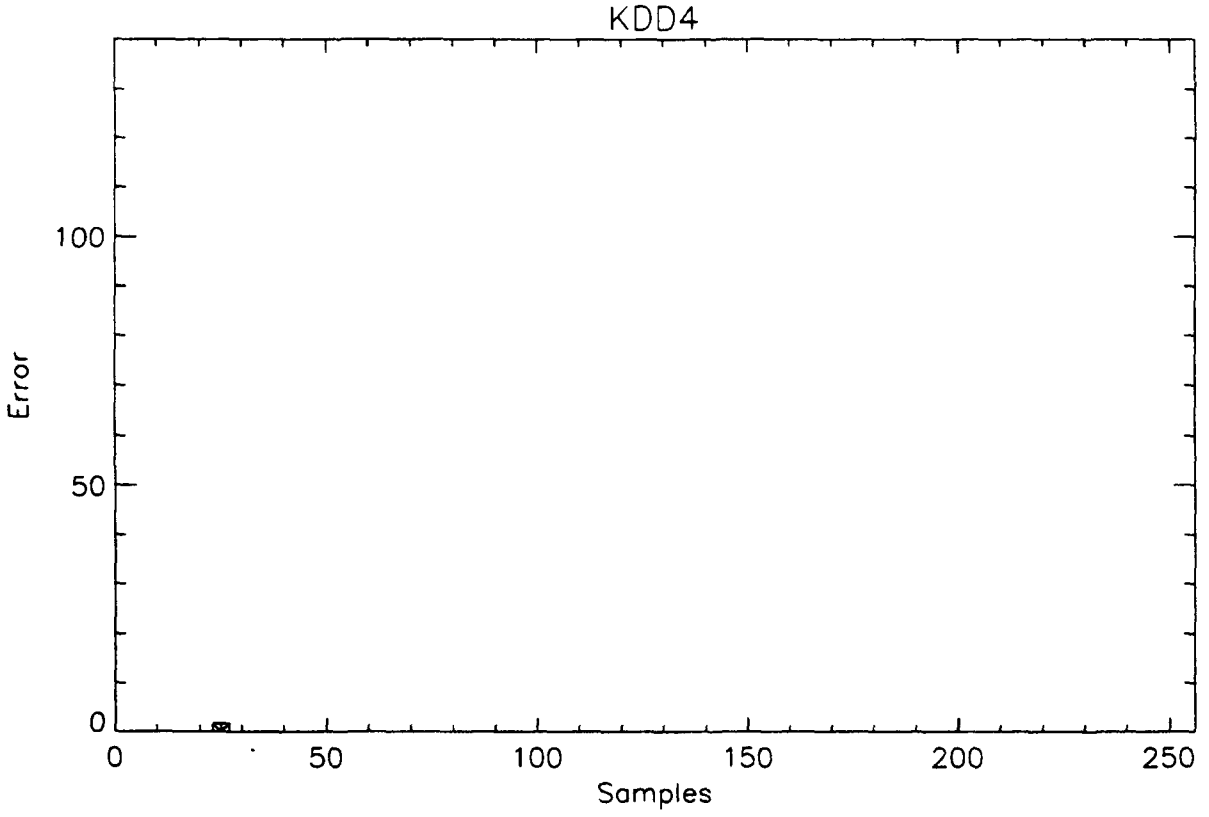


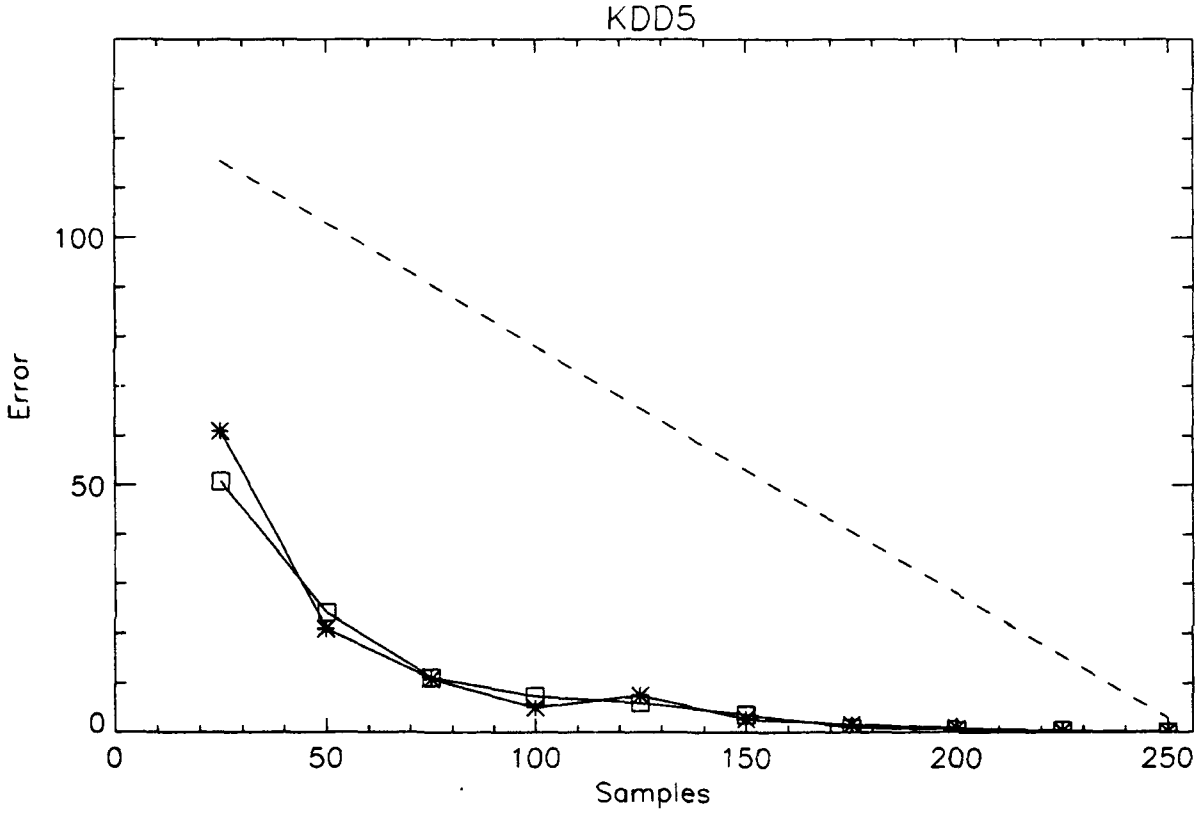
Chance

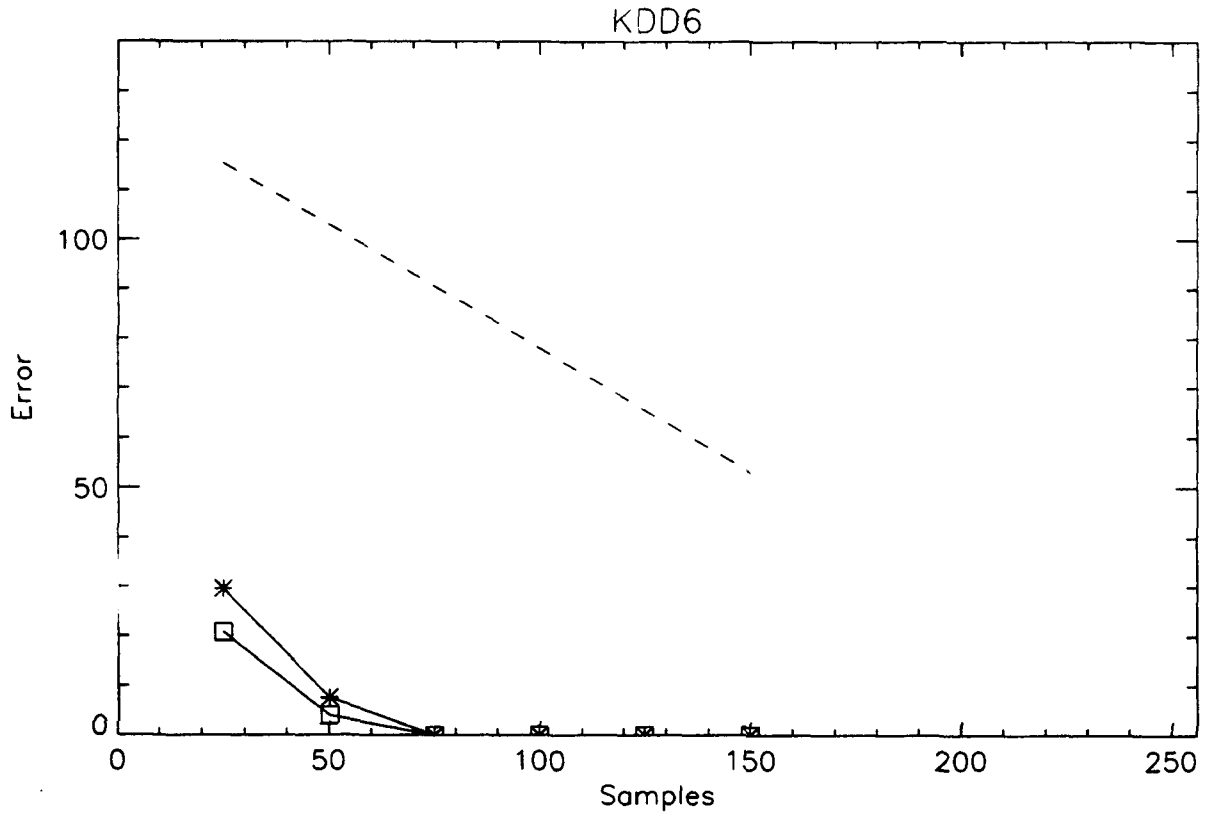
Average Error for FLASH (dni0e300)

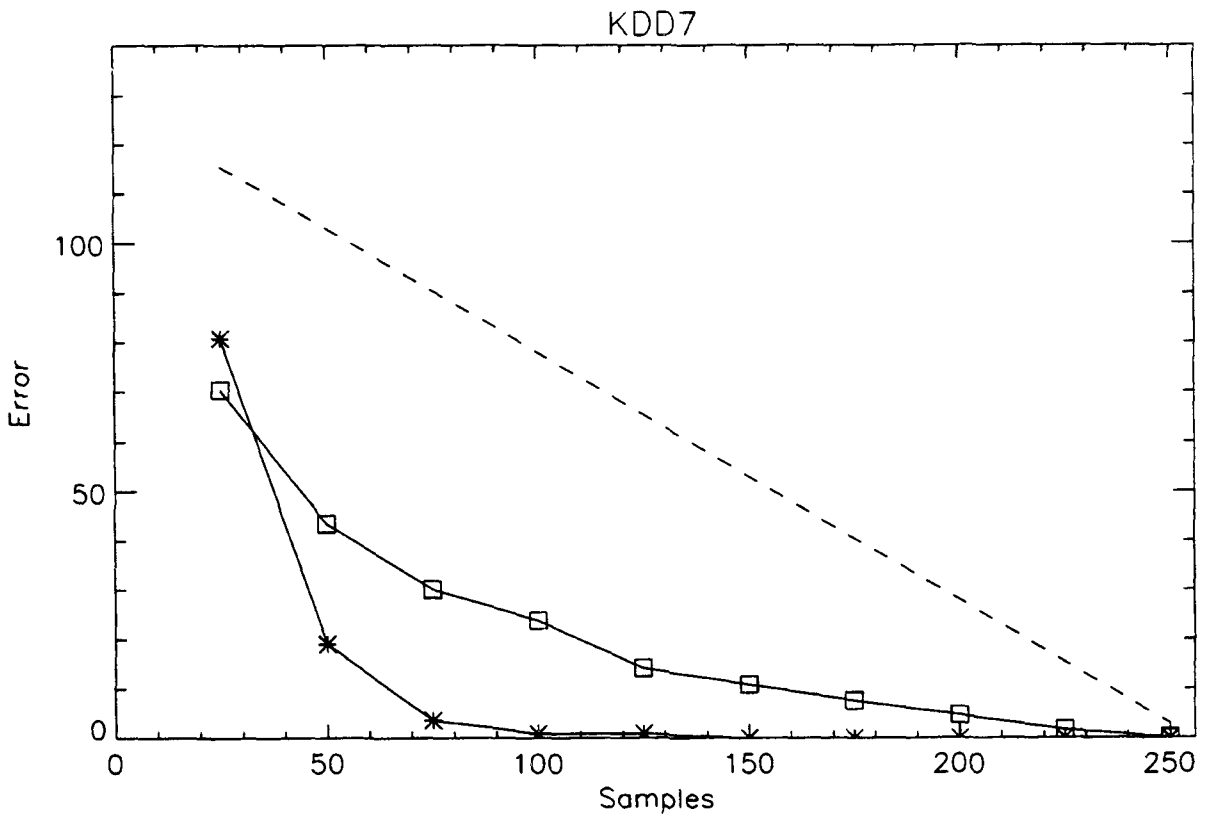
Average Error for C4.5 -m 0 -t 10

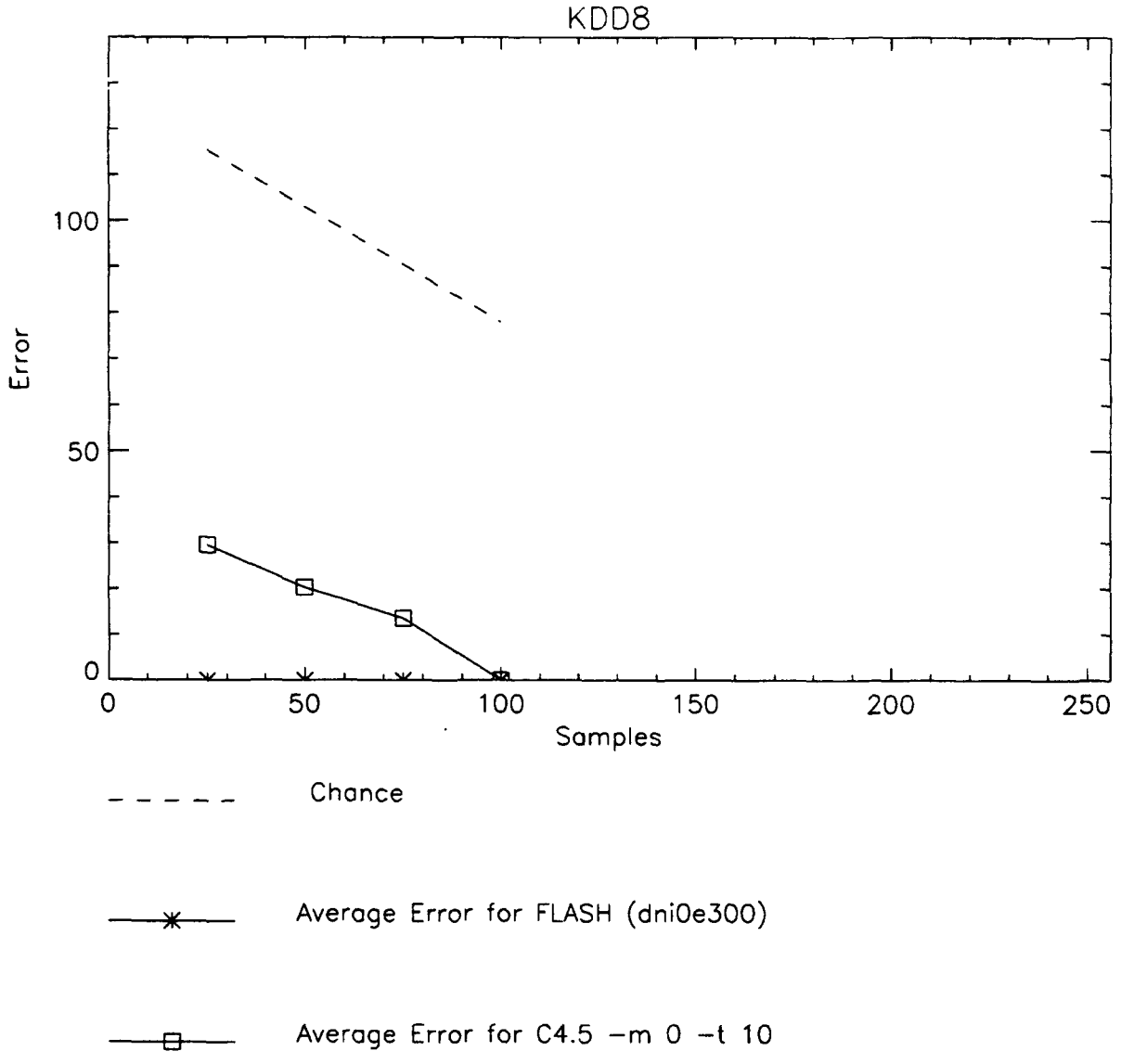

KDD3
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10

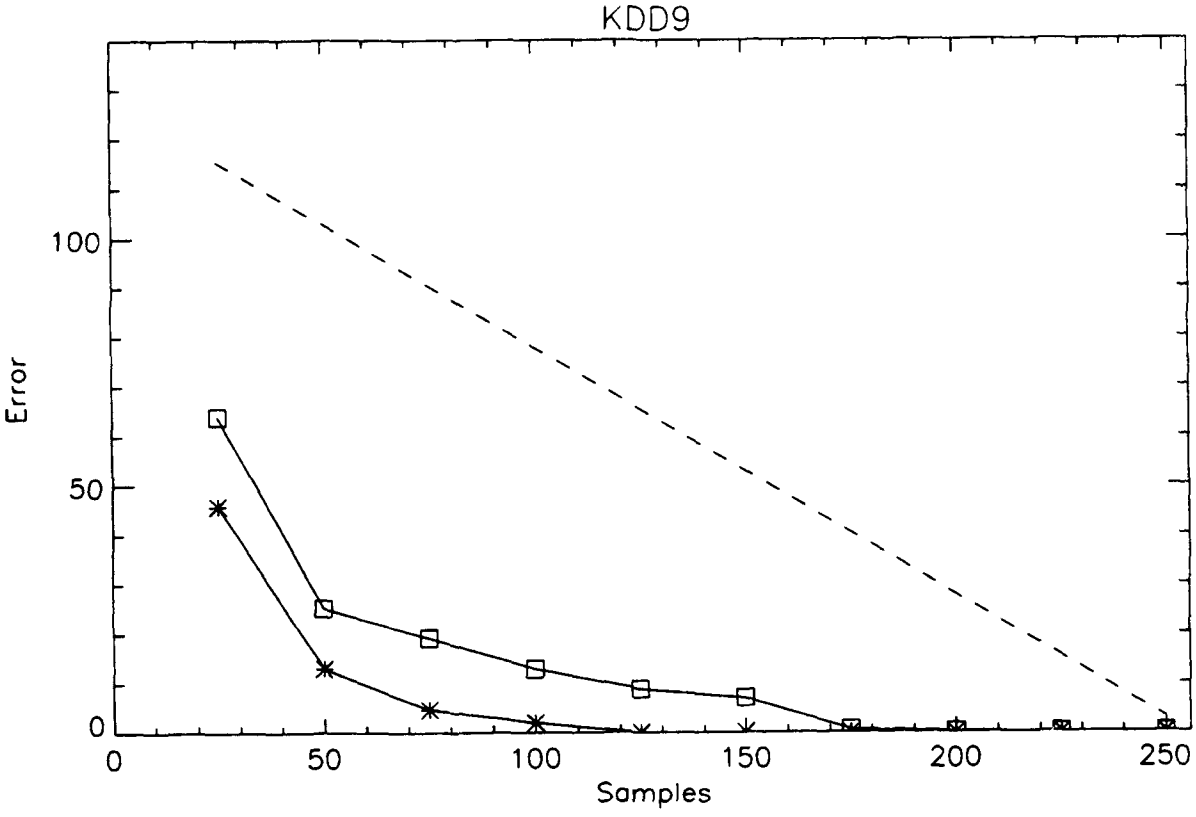

KDD4
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10

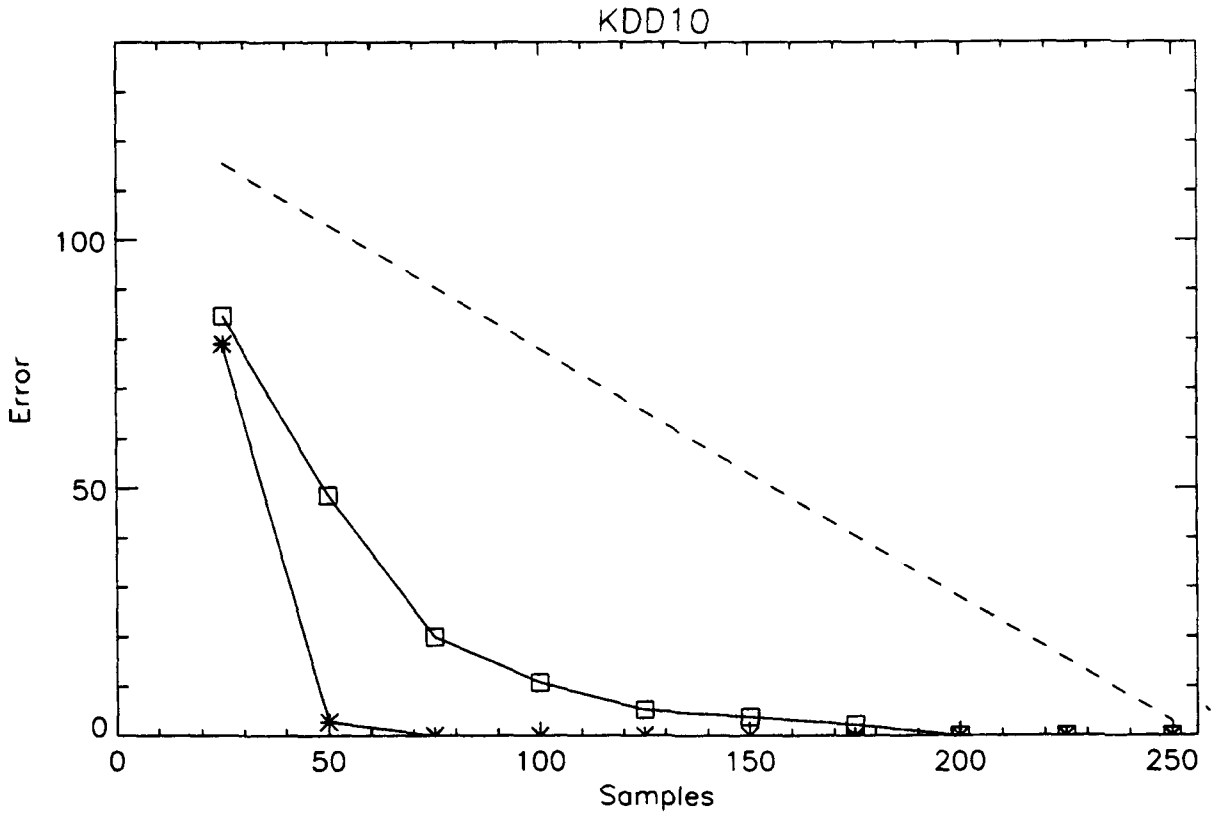

KDD5
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10


KDD6
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10


KDD7
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10


KDD8
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10


KDD9
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10


KDD10
Error
100
50
0
0
50
100
150
200
250
Samples
Chance
Average Error for FLASH (dni0e300)
Average Error for C4.5 -m 0 -t 10

Acknowledgements

I would like to thank all of the members of the (extended) Pattern Theory Team for their helpful suggestions and comments.